

AFRL-IF-RS-TR-2002-268
Final Technical Report
October 2002



PLAN MANAGEMENT CAPABILITIES FOR AUTONOMOUS AGENTS: EXTENDING THE BASIC MECHANISMS

University of Pittsburgh

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. J688

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.


The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2002-268 has been reviewed and is approved for publication.

APPROVED:

A handwritten signature in black ink, appearing to read "Frank H. Born".

FRANK H. BORN
Project Engineer

A handwritten signature in black ink, appearing to read "Eugene C. Blackburn".

FOR THE DIRECTOR:

EUGENE C. BLACKBURN, Chief
Information Technology Division
Information Directorate

REPORT DOCUMENTATION PAGE			<i>Form Approved</i> OMB No. 074-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing this collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE October 2002	3. REPORT TYPE AND DATES COVERED Final May 00 – May 02	
4. TITLE AND SUBTITLE PLAN MANAGEMENT CAPABILITIES FOR AUTONOMOUS AGENTS: EXTENDING THE BASIC MECHANISMS			5. FUNDING NUMBERS C - F30602-00-2-0547 PE - 62301E PR - AGEN TA - T0 WU - P1	
6. AUTHOR(S) Martha E. Pollack				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Pittsburgh Pittsburgh Pennsylvania			8. PERFORMING ORGANIZATION REPORT NUMBER N/A	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Defense Advanced Research Projects Agency AFRL/ITB 3701 North Fairfax Drive 525 Brooks Road Arlington Virginia 22203-1714 Rome New York 13441-4505			10. SPONSORING / MONITORING AGENCY REPORT NUMBER AFRL-IF-RS-TR-2002-268	
11. SUPPLEMENTARY NOTES AFRL Project Engineer: Frank H. Born/ITB/(315) 330-4762/ Frank.Born@rl.af.mil				
12a. DISTRIBUTION / AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.				12b. DISTRIBUTION CODE
13. ABSTRACT (Maximum 200 Words) Software agents can assist a human user in overseeing, managing and coordinating large and potentially complex sets of plans. Such agents are potentially of great value in both military and civilian applications. This report addresses plan management software agent technology. There were three main goals in the research: 1. Design, implementation, and experimental analysis of algorithms for supporting plan management capabilities in agent software systems. 2. Adaptation of the plan management agents into a system that can interact with the agent sandbox, or "grid", developed in the DARPA CoABS program for agent registry, search, and interaction. 3. Development of initial techniques for using the plan management agent algorithms to support outsourcing of tasks in agent communities,				
14. SUBJECT TERMS Planning, Agent Software, Artificial Intelligence				15. NUMBER OF PAGES 87
				16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL	

TABLE OF CONTENTS

<u>SECTION</u>	<u>TITLE</u>	<u>PAGE</u>
1.	Plan Management Capabilities for Autonomous Agents: Extending the Basic Mechanisms.....	1
2.	A Scheme for Integrating E-Services in Establishing Virtual Enterprises.....	6
3.	Flexible Dispatch of Disjunctive Plans.....	15
4.	Flexibility Measures of Sets of Plans.....	21
5.	An Evaluation of the Java-Based Approaches to Web Database Access (2002)	43
6.	An Evaluation of the Java-Based Approaches to Web Database Access (2000)	65
7.	A Survey on the Java-Based Approaches for Web Database Connectivity.....	80

Plan Management Capabilities for Autonomous Agents: Extending the Basic Mechanisms

Prof. Panos Chrysanthis
Computer Science Dept.

University of Pittsburgh
Pittsburgh, PA 15260
(412)624-8924
panos@cs.pitt.edu

Prof. Martha E. Pollack
Dept. of Electrical Engineering and Computer
Science

University of Michigan
Ann Arbor, MI 48103
(734) 615-8048
pollackm@eecs.umich.edu

Technical Results.

This report presents the work conducted under the direction of Prof. Panos Chrysanthis (Univ. of Pittsburgh) and Prof. Martha E. Pollack (Univ. of Michigan) as part of the DARPA Control of Agent-Based Systems Program, on Contract #F30602-00-0547. The contract covered roughly a one-year period, from 5/15/00-6/30/01.

This project was aimed at advancing the state of the art in plan-management agents: agents that assist a human user in overseeing, managing, and coordinating large and potentially complex sets of plans. Such agents are potentially of great value in both military and commercial applications. For example, an officer managing a complex military operation might use a plan management agent to track the allocation of resources such as personnel and equipment, guarantee consistency among various operational goals, and guide reactions to new developments. In this project, we extended an existing prototype plan-management agent (PMA), previously built in our research group. We focused on, and achieved three main goals:

1. The design, implementation, and experimental analysis of more powerful algorithms for supporting plan management capabilities in the PMA (Plan Management Agent). In particular, we developed algorithms for checking feasibility of proposed plan updates that are two orders of magnitude faster than the previous state-of-the-art [8,9]; we constructed the first framework for dispatching plans with rich temporal constraints [2,3]; and we began work to increase the efficiency of assessing plan cost in context [7].
2. The adaptation of the PMA plan management agent to a grid-aware system, along with the analysis of alternative Java-based methods for supporting mobile agents in grid-like systems [4,5,6].
3. The development of initial techniques for using PMA algorithms to support outsourcing in electronic communities [1].

By achieving the first goal, we have made it possible to design and implement individual agents that are better suited to the dynamic, multi-agent environments that are the focus of the CoABS effort. The second goal demonstrated that our prototype system could be integrated into the set of software agents being developed for CoABS; it also led to results about that can impact the design of future tools, like the CoABS grid, for managing suites of agents. Finally, our work on the third goal points the way towards taking greater advantage of the techniques that we initially developed to allow an agent to coordinate its own plans in a dynamic, multi-agent environments: specifically, we show how an agent can use those same techniques to coordinate plans with other agents in the environment.

Below, we provide a very brief discussion of each topic; we have attached our relevant papers, which provide further detail.

Improved Plan Management Support in PMA.

One of the major objectives of this project was to continue the development and refinement of plan-management algorithms. We made major progress in particular in algorithms for checking the feasibility of proposed plan updates. Note that some of our work on this topic was supported under the auspices of our DARPA TASK contract, #F30602-00-2-0621. Here we report on our overall results, obtained by leveraging the work in the two projects.

One of the most efficient ways to achieve plan update is to merge new, partial plans into existing sets of commitments, rather than attempting to replan from scratch. Indeed, plan merging is central to managing plans in dynamic and uncertain environments, because in these environments an agent may adopt a goal for a future activity, form a possibly incomplete plan for it, and then consider, commit to, and plan for additional goals before completing – or possibly even beginning – execution of the first plan. Thus, as time proceeds, the agent is continually forming plans for new goals in the context of its existing goals and plans. A plan-management agent must help a human in updating and maintaining his commitments.

Various approaches could be used to generate plans in the context of prior plans. A simple approach would rely on traditional AI methods of planning for conjunctive goals. With this approach, whenever the agent encountered a new goal G in a setting in which it already held goals G_1, \dots, G_n , it would form a new planning problem for the conjoined goal $G_1 \wedge \dots \wedge G_n \wedge G$. However, this approach has at serious problems—including computational inefficiency that results from the failure to reuse previous computation. We have thus been investigating an alternative approach, which involves holding fixed the plans for G_1, \dots, G_n , and merging into them a new plan for G . In our previous work, we developed an algorithm and implemented system for plan merging that was much more powerful than previous approaches in that it allowed for plans with conditional branches and a certain class of temporal constraints, namely constraints that can be represented as simple temporal problems (STPs). The basic idea of our earlier approach was to solve two constraint-satisfaction processing problems (CSPs) in an interleaved fashion: the first modeled the plans being merged, and identified potential conflicts between them, suggesting a candidate resolution, while the second was a temporal CSP that was used to determine whether the suggested resolution was in fact temporally consistent.

We designed and fully implemented a new system for plan merging, called Epilitis. Using Epilitis, we can now perform plan merging on an even wider set of plans, allowing arbitrary disjunctive temporal plans. Not only does this increase the expressive power of the approach, but, surprisingly, it can have a potentially beneficial effect on plan merging, because typically

there are several alternative candidate resolutions to any conflict. For example, if one activity should be performed sometime Friday between 9am and 5pm, for a one to two hour duration, and there is already a commitment to attend a meeting from 11am to noon, then the way to resolve the conflict is to ensure that the first activity is begun either at 9 or sometime after noon. Such resolutions—essentially additional constraints on the merged plan—need not be binary: there may be many alternatives. In our previous approach, there was a need to jump between the two CSP problems whenever one alternative failed. However, because we can now directly model and reason about disjunctive constraints, we maintain and solve only a single CSP. Details can be found in [8,9].

Once Epilitis was designed, we were able to conduct a thorough investigation of techniques for speeding up its processing. Because Epilitis is performing constraint-satisfaction processing, we were able to draw on a number of methods for CSP efficiency. More specifically, Epilitis is solving a Disjunctive Temporal Problem (DTP), and we thus carefully analyzed the techniques used in the prior DTP-solving work, determining which ones we most effective and then analyzing their interactions to enable us to combine the best techniques. Integration of these techniques was a challenging technical problem. We also added no-good learning, a pruning strategy that had not previously been used in DTP-solving. Experimentation showed that no-good learning was a particularly powerful tool for speed-up in plan-merging problems. As a result of this work, we were able to achieve a two order-of-magnitude speed-up with Epilitis as compared to the previous state-of-the-art DTP solver, TSAT. Again, more details of this work can be found in [8,9].

In addition to our work on plan merging, we also investigated the question of how to dispatch plans expressed as DTPs. Many agent systems must perform both planning and execution: they include a plan deliberation component to produce plans that are then dispatched to an execution component, or *executive*, which is responsible for the performance of the actions in the plan. When plans have temporal constraints, dispatch may be non-trivial, and the system may include a distinct *dispatcher*, which is responsible for ensuring that all temporal constraints are satisfied by the executive. Previous research on dispatch has been restricted to plans with simple, nondisjunctive temporal constraints. But plan-management systems dealing with plans represented as DTPs must also be able to perform dispatch of plans with disjunctive constraints. (Such systems do not, of course, executing the dispatched plans itself, but they must still be able to perform dispatch so that they can provide appropriately timed reminders to the user.) We identified four key algorithmic and complexity properties that must be satisfied by a dispatch algorithm, and developed an algorithm for DTP dispatch that has these properties. See [2,3].

PMA for Mobile Agents.

A second major goal of this project was to develop a grid-aware version of PMA. We successfully completed this task and reported on it, providing a demo, at the Aug. 2000 CoABS PI meeting. In follow-on work, we developed a web-based version of PMA, which allows for anywhere-access of the system. Access to the system is available from Prof. Panos Chrysanthis

As part of this task, we also conducted a detailed experimental evaluation of alternative Java-based methods for remote access to databases, such as the PMA plan database. The study considered both RPC and non-RPC approaches. The evaluation was aimed at comparing both performance (response time under different loads) and programmability (number of system calls at the client and the server site). The results of these studies can have significant potential impact on the design of advanced grid-like systems. The primary results are (1) best performance is not

always achievable with high programmability and low resource requirements, and (2) the mobile agent technology needs to improve its programmability while giving particular emphasis in its infrastructure. Further details, including quantification of these results, can be found in [4,5,6].

Outsourcing in Agent Systems

The third component of the research addressed the question of how plan-management techniques developed for PMA could be used to support outsourcing in electronic communities. More specifically, we developed a novel approach to combining plan-management techniques such as plan merging with workflow mechanisms, to enable automated outsourcing of tasks by individual agents in an electronic community. Our method takes two workflow views, one representing a service request and the other a service provision (an advertisement), with a mix of vital and nonvital steps and a rich set of constraints, and returns a list of possible legal combinations, if any exist. It then uses plan-merging techniques to find potential conflicts between the two workflows, and to suggest additional constraints that can resolve the conflicts. In particular, we make use of the DTP solving algorithm mentioned above to guarantee local as well as global plan consistency. The solutions returned by the DTP-solver represent terms for the establishment of a new Virtual Enterprise (VE), and can be evaluated by each side to determine which is most desirable. To date, we have designed and provided specifications for such a system; full implementation and experimental evaluation will be conducted in future research. See [1] for more discussion.

Publications supported by this Contract

1. A. Berfield, P. Chrysanthis, I. Tsamardinos, M. E. Pollack, and S. Banerjee, "A Scheme for Integrating e-Services in Establishing Virtual Enterprises," to appear in *12th IEEE Workshop on Research Issues in Data Engineering*, Feb. 2002.
2. I. Tsamardinos, M. E. Pollack, and P. Ganchev, "Flexible Dispatch of Disjunctive Plans," 6th European Conference on Planning, Oct. 2001.
3. P. Ganchev, "Flexibility Measures of Sets of Plans," University of Pittsburgh Intelligent Systems Program M.S. Project Report, June, 2001.
4. S. Papastavrou, P. K. Chrysanthis, G. Samaras, and E. Pitoura. "An Evaluation of the Java-based Approaches to Web Database Access," to appear in *International Journal of Cooperative Information Systems*, 2002.
5. S. Papastavrou, P. K. Chrysanthis, G. Samaras, and E. Pitoura. "An Evaluation of the Java-based Approaches to Web Database Access," *Proc. of IFCIS/VLDB Cooperative Information Systems*, Sept. 2000, pp. 102-113.
6. S. Papastavrou, P. K. Chrysanthis, G. Samaras, E. Pitoura. "A Survey of the Java based Approaches for Web Database Access." *Proc. of the 10th IEEE Mediterranean Electrotechnical Conference*, May 2000.
7. S. Ramakrishnan, "Cost Assessment in Disjunctive Temporal Plans," Technical Report, University of Michigan, 2001.

8. I. Tsamardinos, "Constraint-Based Temporal Reasoning Algorithms, with Applications to Planning," University of Pittsburgh Intelligent Systems Program Ph.D. dissertation, Aug. 2001.
9. I. Tsamardinos and M. E. Pollack, "Efficient Solution Techniques for Disjunctive Temporal Problems," in preparation.

A Scheme for Integrating E-Services in Establishing Virtual Enterprises*

Alan Berfield, Panos K. Chrysanthis
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
{alandale,panos}@cs.pitt.edu

Martha E. Pollack
Department of EE and Computer Science
University of Michigan
Ann Arbor, MI 48109, USA
pollackm@eecs.umich.edu

Ioannis Tsamardinos
Intelligent Systems Program
University of Pittsburgh
Pittsburgh, PA 15260, USA
tsamard@cs.pitt.edu

Sujata Banerjee[‡]
Info. Sci. & Telecom. Dept.
University of Pittsburgh
Pittsburgh, PA 15260, USA
sujata@tele.pitt.edu

Abstract

An important aspect of Business to Business E-Commerce is the agile Virtual Enterprise (VE). VEs are established when existing enterprises dynamically form temporary alliances, joining their business in order to share their costs, skills and resources in supporting certain activities. Currently, existing enterprises use workflows to automate their operation and integrate their information systems and human resources. Thus, the establishment of a VE has been viewed as a problem of dynamically expanding and integrating workflows. In this paper, we present an approach to combining workflows from different enterprises, using techniques developed in the Artificial Intelligence literature on planning. Our method takes two workflow views, one representing a service request and the other a service provision (advertisement), with a mix of vital and nonvital steps and a rich set of constraints, and returns a list of possible legal combinations, if any exist. It then uses plan-merging techniques to find potential conflicts between the two workflows, and to suggest additional constraints that can resolve the conflicts. The returned solutions represent terms for the establishment of a new VE, and can be evaluated by each side to determine which is most desirable.

1. Introduction and Motivation

Electronic Commerce is expanding from the simple notion of E-Store to the notion of *Virtual Enterprises* (VEs) where existing enterprises dynamically form temporary alliances, joining their business in order to share their costs, skills and resources in supporting certain activities. An example of a VE in the context of the travel industry would be the collaboration of different travel agents, airlines, ground transportation services, hotels, restaurants and entertainment services in order to set up and manage a tourism trip.

Many enterprises use workflows to automate their operation and integrate their information systems and human resources [19]. A workflow consists of a set of *activities* (also called *tasks*) that need to be executed according to given temporal constraints over a combination of heterogeneous database systems and legacy systems. A major challenge has been the development of *workflow management* systems (e.g., [9, 5, 13, 1]). Several techniques have been developed for correct and reliable specification, execution, and monitoring of workflows and the involved external support. Many of these techniques are extensions of those in transaction processing in databases combined with general middleware services such as those found in CORBA/DCOM and more recently in Java-based services such as Jini from Sun and E*Speak from HP.

Very recently the idea of the use of workflows to support multi-organizational processes that form a virtual enterprise has attracted some attention [10, 6, 8]. The establishment of a VE can be seen as a problem of dynamically expanding and integrating workflows in decentralized, autonomous and interacting workflow management systems

*This material is based upon work partially supported by NSF IIS-9812532 and AFOSR F30602-00-0547 awards.

[‡]On Leave of Absence at Hewlett-Packard Laboratories, MS 1U-17, Palo Alto, CA 94304, USA.

[2, 7, 12]. During the establishment of a VE, a distributed, multi-organizational workflow emerges from the dynamic merging and reconfiguration of workflows representing E-Services in the participating enterprises. In our previous work, we looked at using mobile agents as a platform for advertising, negotiating, and exchanging control information about E-Services for the establishment of VE's [6]. In this paper, we focus on a method for verifying that the VE is compatible with workflows in the participating enterprises.

The contribution of this paper is a new method for establishing VEs, involving both the generation of outsourcing requests and the validation of constraints. The scheme incorporates techniques developed in the Artificial Intelligence (AI) literature on planning, specifically algorithms for merging temporal plans. Within the AI literature, a plan is a collection of steps (i.e., tasks), with causal, temporal, and resource constraints. A plan is intended to represent a course of action that will achieve a specified goal when executed beginning in a specified initial state. Critical to the notion of plans is that of *causal structure*: the steps in each plan are specified in terms of their preconditions and effects, and the plan records information about which steps cause (or *establish*) the preconditions of other steps. When merging together two plans, it is necessary not only to check that there are no violations of the temporal and resource constraints of the plans being merged, but also to ensure that the necessary causal relations are maintained in the merged plan. We argue in this paper that similar consistency requirements also hold when a VE is formed.

In the next section, we review the basic structures used in workflows. In particular, we describe a class of workflows that include specifications of preconditions and effects. In Section 3, we describe a VE and its components. Section 4 describes our detailed scheme for establishing such a VE. Section 5 deals with the current state of our implementation. We conclude with a summary in Section 6.

2. Workflow Model

Workflows encode tasks and the relationships among them. Workflow specification formalisms generally provide a small set of basic control flow relationships among tasks. Typically there are four such relationships: OR-split, AND-split, OR-join and AND-join. The first two relationships are used to specify branching decisions in a workflow whereas the remaining two specify points where activities converge to initiate the next activity within a workflow. An OR-join specifies alternatives whereas AND-join specifies required activities.

While the relations just listed provide information about the relative ordering of the tasks in a given workflow, to handle the problem of forming Virtual Enterprises, it is also necessary for the workflow to model a significant amount

of information about each task. Thus, we will assume an enriched model of workflows in which each task has the following information associated with it:

- *Pre- and postconditions*, which specify what must be true before a task can be executed, and what will be made true as a result of the task's performance.
- *Causal links*, which relate each task that establishes a condition (listed in its postconditions) to the task that requires it (listed in its preconditions).
- *In- and out-parameters*, which are used in the evaluation of preconditions and postconditions. They carry information and engender data flow during execution. For example, a credit card number could be an in-parameter to a "pay for dinner" task.
- *Temporal Constraints* that specify the earliest and latest start and end times of a task, as well as the minimal and maximal durations of the task. They can be absolute times or relative to the execution of other tasks.
- *Resource Constraints*, which specify the equipment, material, or agent resources required for the task.
- *Significance*, which indicates whether the task is *vital* to the workflow and therefore must be executed, or whether it is *nonvital*, and need only be executed if feasible [6].
- *Cost*, which represents the price of the task.

Other information may also be associated with each task, such as rules for exception handling should the task fail. However, we will not be concerned with these types of information in the current paper.

As shown in Figure 1, a workflow can be graphically depicted with nodes (thick boxes) denoting activities and arrows denoting precedence. The figure represents a business trip from [15]. Shaded nodes indicate vital activities that must be completed to ensure proper execution. Nodes with a pair of dashed lines leading to another workflow are hierarchical activities: those that can be decomposed into workflows themselves. AND-splits and AND-joins are represented implicitly when two or more causal links emanate or arrive at a node respectively. To represent OR-splits we insert a conditional node that creates two new execution contexts (branches), e.g., one for success and one for failure (in Figure 1, these are shown as nodes with edges labeled S and F). Tasks are executed only when their context is true. The OR-join is represented implicitly when the context S or F disappears from the labels of subsequent edges.

We are assuming a typical Workflow Management System (WfMS) architecture with our enriched model of workflows. Specifically, a WfMS consists of the following three basic components:

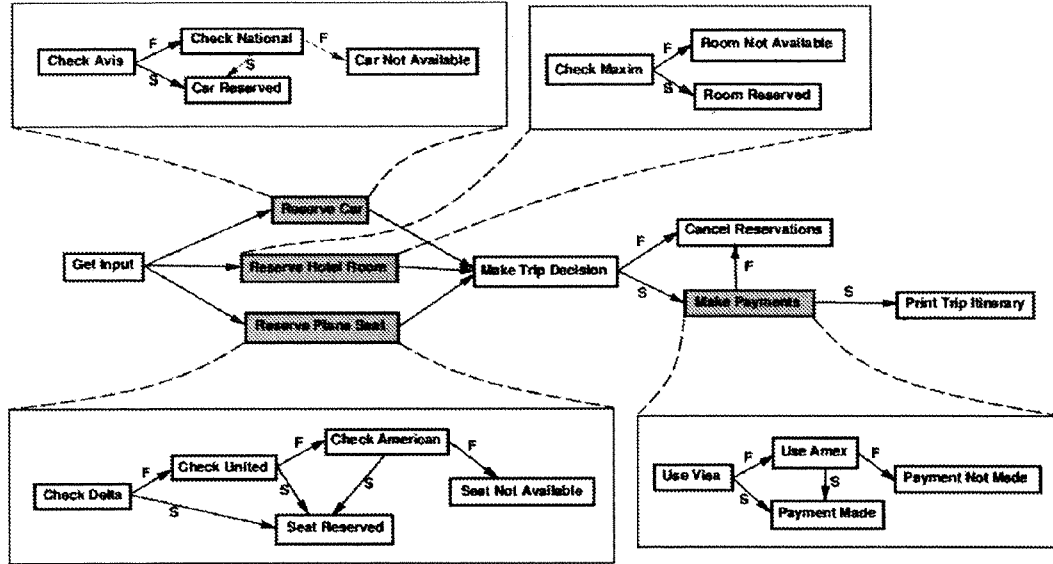


Figure 1. Trip Plan Workflow

- *Workflow Schema Library*, which contains workflow schemas or templates and generic constraints.
- *WfMS Services*, functions provided by the WfMS for managing workflows. These include specifying workflows, verifying their correctness, instantiating and scheduling them, executing them, and monitoring their execution.
- *Workflow Repository*, which contains all instantiated and scheduled workflows, i.e., the workflows the business is committed to performing.

3. Forming Virtual Enterprises

A Virtual Enterprise (VE) is formed when a business decides to commit to a new workflow, while outsourcing some of the work involved in that workflow. Consider the example of Jane Smith, an executive planning a trip to Vienna. She gets in touch with a travel agency to arrange the trip. She decides that while she is there she would like to attend an opera and tour the Art Museum. This adds the nonvital nodes “buy opera ticket” and “buy museum tour ticket” to the trip schema (Figure 1). The travel agency lacks connections with the entertainment/opera industry, so is unable to purchase such tickets. In order to satisfy the customer, they decide to outsource those tasks.

The above example represents a common reason for outsourcing. When a business receives a new request from a client, it takes the form of an instantiated workflow schema from its Workflow Schema Library. The client may have added constraints and/or customized the schema by adding

new nodes, which could represent extra or special activities and opportunities. The business may select some of the tasks from the workflow to outsource and/or it may select some of the open conditions from the workflow and outsource their achievement. This outsourcing establishes a VE.

In our VE workflow specification, we use the notion of views to express outsourcing. Any subgraph of a workflow graph defines a *segment* or a *view* of the workflow. Formally, a workflow view can be defined as a *projection* on the graph based on some criteria ($projection(workflow, < criteria >)$). For example, consider the view that includes all and only the vital nodes of the full workflow. The requirement that nodes be vital is the criteria used by the projection.

$$VitalView = projection(workflow, \{a \mid a \in workflow \wedge a.significance = vital\})$$

The nodes in a view retain all information of their originals, including all constraints. However, because all constraints are maintained, a view may have nodes that have temporal constraints referring to other nodes not actually in the view, and may also have broken causal links possibly resulting in unsatisfied preconditions (i.e., a node in the view could have a precondition that was established by some node in the full workflow that is not in the view).

A workflow view can represent any activity performed by a *service provider* on behalf of a *service requester*. Consequently, workflow views can be used to express service requests or service provision (advertisement). In our proposed system, it is these workflow views that are being requested and advertised.

In our scheme, a request has the following structure:

Requests: $Rq = (P, G, RW)$

where P = Service Requester Profile,

G = set of Goals,

RW = Requested Workflow View

The profile can contain various information about the requester, such as name, site identification, credentials, etc.. It may also contain a target price range. The set of goals is a list of all goals (postconditions) that need to be accomplished. The workflow view captures all temporal constraints and resource usage issues involved.

In the example above, the request includes the profile of the travel agency, the goals "opera ticket purchased" and "museum ticket purchased", and a view with two nodes that indicate times by which the tickets must be purchased.

A requested workflow view can be potentially augmented during negotiation to match the service provider's workflow, reflecting opportunities, omitted activities and data. During a negotiation we may decompose the required view into several views and seek other service providers for the other parts of the view. In this way, a single initial request may lead to the establishment of a VE comprising multiple enterprises. A VE comprising multiple enterprises can also result when a service provider's view includes outsourcing. We will elaborate on this in the next section.

The structure of an advertisement is the same as that of a request.

Advertisements: $Ad = (P, G, AW)$

where P = Service Provider Profile,

G = set of Goals,

AW = Advertised Workflow View

It includes a profile, the set of goals accomplished, and a workflow view encompassing constraints. The profile, in addition to other information, may contain cost information for the workflow as a whole, such as minimum cost, maximum cost (cost with all nonvital steps), or both. The list of goals indicates what the advertised workflow actually does, and may also include goals associated with nonvital activities. Such advertisements will typically be stored in the databases of trading servers. Each provider may have a set of advertisements with the same goals but with a different associated workflow view (i.e., different constraints).

To return to our example, an advertisement that would be of interest to the travel agency would be for a business that specializes in Vienna cultural events, including opera. The single goal "opera ticket purchased" is accomplished. Its workflow view includes the tasks "contact opera houses", "read current reviews", and "purchase ticket."

The VE environment is a distributed environment. It consists of multiple businesses, acting as requesters and providers, using services provided by negotiation areas or trading places. The trading places could contain databases

of advertisements and could provide services allowing businesses to both place advertisements and to find advertisements that meet their goals. Standardization of representation is clearly required (particularly of preconditions, effects, and goals), and could be enforced by the trading servers. A portion of this environment is shown in Figure 2. Two negotiation areas are depicted, as well as four businesses' WfMS. Shaded nodes again represent vital activities, and an advertised hierarchical Opera activity is shown partially expanded.

3.1. Commitment and Outsourcing Request Generation

In order to commit a new, possibly customized workflow, a WfMS needs to make sure that it is schedulable. A workflow is schedulable if it is *correct*, *complete*, and *compatible* with existing commitments.

Definition 1 Workflow Correctness: A workflow is correct if and only if

1. it has no conflicting temporal or resource constraints,
2. for each goal/precondition P , there is a task that achieves P (the producer task), and it is ordered before the task that requires it (the consumer task), and
3. for each goal/precondition P , no task that may negate P can possibly be ordered in between the producer and the consumer.

This notion of correctness is important as only correct workflows can possibly be executed. Note that some workflows may contain preconditions that are assumed to be established independently of the workflow itself. We will call such preconditions *open* with respect to the workflow. A simple example of such an open precondition is a workflow for renting a car that assumes the precondition of having a driver's license. Workflows with such open preconditions are incorrect until they have been combined with other workflows that establish all open preconditions.

Definition 2 Workflow Completeness: A complete workflow is a workflow that specifies all tasks needed to achieve its goals and preconditions.

Definition 3 Workflow Compatibility: A workflow is compatible with another if none of its nodes conflict with any of the other's (and vice versa).

This means that the temporal constraints, resource usage, and postconditions of its nodes do not prevent the execution of the nodes in the other workflow (though they may place limits on when those nodes can be executed). So for example, a compatibility conflict between workflows arises if

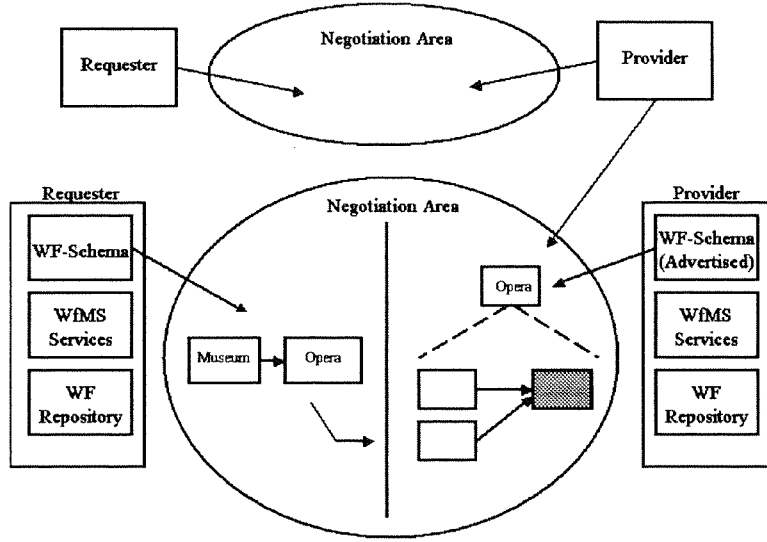


Figure 2. VE Environment

two tasks that use the same resource (e.g., equipment) are set to execute at the same time. Another example is a task that dictates that a robot move to the printing room for the purpose of getting a faxed itinerary, which conflicts with a task that moves the robot to another room that could be executed after going to the printing room but before fetching the fax.

An alternative definition of the compatibility of two workflows is that the workflow resulting from their union is correct. We propose the notion of a *merge* with the Workflow Repository for determining the compatibility of a workflow with the currently scheduled workflows (in the Repository). If the merge is successful, the new workflow can be committed and its execution enabled. If the merge is unsuccessful, the new workflow is not compatible and the business may consider outsourcing.

An effective merging process will check whether the above requirements (correct, complete, and compatible) are met, and will indicate where problems lie: what nodes are conflicting with others, which have unsatisfied (open) preconditions, or which the business lacks the necessary expertise (i.e., roles as resources) to accomplish. It may also suggest additional temporal or resource constraints that are required to ensure that they are met. However, it's desirable to impose a minimal set of extra constraints, i.e., to provide a least-commitment response, as this allows increased flexibility to respond to changes that may arise during execution.

The merge process can also be used to identify and construct outsourcing requests. In the event of an unsuccessful merge, any nodes from the new view that are indicated as problems by the merging process (those having irresolvable

resource conflicts with existing commitments) will form part of the requested workflow view VR_q by extracting them from the full workflow using projection. In addition to these nodes, for each open precondition in the new view not satisfied by the existing commitments (such preconditions will be found by the merge process), a new place-holder node is added to the view. Each of these new nodes represents a task that accomplishes one of the open preconditions, i.e., it has one of the open preconditions set as its postcondition, and any associated temporal and causal links are applied. The complete set of postconditions of every node in VR_q make up the goal set of the request, G . In the simplest case VR_q would be a single node, with associated constraints. More complex cases would involve multiple nodes and richer constraints.

Recall that projected nodes maintain all constraints and conditions they had in the parent workflow, and may therefore include unsatisfied preconditions and temporal constraints referring to nodes not in the view. This is not really a problem as they will be satisfied by non-outsourced nodes. The preconditions, along with in-parameters, represent the input to the outsourced view. Goals and out-parameters of the outsourced nodes represent the output.

4. Outsourcing Scheme

In this section, we discuss in detail the steps for outsourcing and establishing a VE.

Let R be a Requester and P be a set of providers $\{P_1, \dots, P_n\}$. R has a set of workflows to which it is already committed, and which it stores in the WF Repository;

let us call them *CR* (commitment workflows at requester). Similarly, each P_i has a set of workflows already committed to; let us call them $CP(i)$.

Let $Rq = (R, G, VRq)$ be a request of R for outsourcing with goals $G = \{G_1, \dots, G_n\}$ and workflow view VRq . Each P_i can provide a set of alternative workflow views $A(P_i)$ for achieving one or more G_j of Rq .

The problem of outsourcing is how to pick a set of workflow views S from the $A(P_i)$ of one or more P_i so that the combined set satisfies Rq and merges with CR , and each $A(P_i)$ in it merges with its provider's $CP(i)$. Specifically, such a set achieves all goals of the outsourced workflow, all temporal constraints are satisfiable, there are no resource conflicts, and for every precondition of every workflow activity in CR and $CP(i)$ there exists a causal dependency that ensures that the precondition will be met.

Formally, we want a set $S = wf_1 \cup wf_2 \cup wf_3 \cup \dots \cup wf_n$, where $wf_j \in \bigcup_i A(P_i)$, $j = 1, \dots, n$ such that

- $postconditions(S) \supseteq G$
(all outsourcing goals are met)
- $\forall wf_x \in S \text{ } postconditions(wf_x) \cap G \neq \phi$
(each workflow achieves at least one goal)
- $compatible(S, CR)$
(S is compatible with the requester's commitments)
- $\forall wf_x \in A(P_y) \text{ } compatible(wf_x, CP(y))$
(each alternative workflow is compatible with its provider's commitments)

The above suggests a solution that has three phases:

1. Finding a set of alternative workflows that satisfy Rq
(*Terms for the Establishment of a VE*)
2. Checking for the satisfaction of $CP(i)$
(*Providers Validation of Terms and E-Service Bids*)
3. Check for the satisfaction of R
(*E-Service Bid Evaluation*)

We elaborate on these phases in the next subsections.

4.1. Phase 1: Terms for the Establishment of a VE

As mentioned previously, we assume in this paper that finding alternative workflow views that satisfy a request Rq is a service provided by trading servers. Each alternative view represents a term for the establishment of a VE. During this first phase the sets $A(P_i)$ of alternative workflow views are generated. These views accomplish the goals G of Rq while not violating any of its constraints. For the sake of simplicity, we will assume in the rest of our discussion that there is only one trading server.

The service searches the database of the trading server, looking for advertisements that meet some or all of the requested goals. Which advertisements are examined first depends on the selection conditions being used. One such

condition would include the desirability of first considering those that accomplish all goals, and only considering multiple, partial matches when all such are found. For each advertisement found and selected, the server must finally determine if it or any of its alternatives (involving different combinations of nonvital nodes) can meet the constraints of the request. This process continues until all advertisements that meet any goals have been examined, or some termination criteria are met (such as a deadline for search time).

For the detailed explanation, we will consider one such advertisement found and selected by the trading server that accomplishes all goals in G ; let us call it $Ad1$.

As shown in Figure 3, the service must determine if $Ad1$ will satisfy the constraints in the request's workflow view Rq . To do this, $Ad1$ and Rq are first stripped down to only vital nodes using projection. Temporal constraints of the vital nodes may need to be adjusted, as any referring to non-vital nodes will be invalid. For any node that has such a constraint, there are four possible situations:

1. The nonvital node referred to has no constraints on its time¹: the constraint on the vital node can be dropped.
2. The nonvital node has an absolute time: that time can be used.
3. The nonvital node has a time relative to some other vital node: the reference to that node can be used.
4. The nonvital node has a time relative to some other nonvital node: that nonvital must be searched in the same fashion for a time or vital node reference.

It may be beneficial to instead store such alternative constraints with the vital nodes in order to save computational time, though the number of nonvital nodes is likely to be small.

Next the service attempts to bind the constraints of the vital-only view of Rq (called RqV in the figure) to the stripped view of $Ad1$ (AdV in the figure). Binding adds the constraints of the requested nodes to the corresponding nodes in the advertisement (those that have the same post-conditions). If AdV cannot support the added constraints (because they conflict with existing ones), the bind fails and the function must backtrack to find a different advertisement. Otherwise, the new bound advertised view BV is added to $A(P_i)$, where P_i is the provider of BV , and the search continues for its variants that include nonvital nodes.

The search for variants of BV considers combinations of BV and nonvital nodes from the full $Ad1$ and Rq . This can be achieved by the function *addNodes*, that adds a group of nodes to the workflow BV , restoring any modified temporal constraints that referred to them. This is basically a merge process. The *addNodes* function fails and returns null if the

¹ By "time" we mean either start or end time of the task, depending on which the specific temporal constraint refers to.

resulting view is incorrect (i.e., the new nodes cannot be added without violating constraints). If the function does not fail, the resulting view is added to $A(P_i)$.

It is interesting to point out that there is another possible method for this search: working in the other direction, starting by adding back all nonvitals and then removing them to find correct alternatives. It is not clear which approach is better, but we intend to investigate this in future work.

In either case, the search will proceed until all possible combinations have been attempted or some other termination criteria have been reached. As most views are expected to have 3 or fewer nonvital steps, finding all possible combinations is not likely to be impractical. Once the search has finished, $A(P_i)$ contains every alternative workflow solution for the workflow $Ad1$.

The service generates a set $A(P_i)$ for each P_i with at least one selected advertisement. The $A(P_i)$'s created in this fashion are now sent out to their respective provider for validation.

4.2. Phase 2: Providers Validation of Terms and E-Service Bids

The second phase of the outsourcing takes place at the providers of the advertisements. Each P_i receives the $A(P_i)$ generated for it in the previous phase, and must determine whether any of the workflow views in $A(P_i)$ are compatible with its $CP(i)$. Each alternative basically represents a potential new incoming workflow to be scheduled. Recall that such scheduling can be accomplished using the merge process. Thus, the provider attempts to merge each alternative with $CP(i)$ independently. Any that fail are removed from $A(P_i)$. Those that succeed can be kept to form the basis for the service bid. Of course, if $A(P_i)$ is empty at the end of this phase, then none of the views were compatible with the provider's commitments.

To generate the full service bid, each view remaining in $A(P_i)$ could possibly be expanded into multiple views if the provider wishes to add additional nonvital nodes (representing special offers or bonuses). Note that such additions would likely increase cost, but would possibly also increase value. The provider may also rank the solutions in order of preference or cost to help the later decision process.

Each provider sends its service bid to the requester to be evaluated in the next phase.

4.3. Phase 3: E-Service Bid Evaluation

In the third phase, the requester evaluates and selects an E-Service bid. Of the views in the service bids returned by the providers, the requester must determine which are compatible with its CR . This is done in exactly the same fashion as with the providers.

Each service bid in the returned list is combined with the rest of the original workflow to form a complete solution. For each solution, a merge is attempted with the committed workflows. Any that fail to merge are discarded. Those that successfully merge are correct views that each accomplish the outsourcing and that are compatible with both the provider's and requester's previous commitments.

The requester may then evaluate these remaining views to make a final decision as to which one will be used, which likely involves cost comparisons.

4.4. Multiple Partial-Solution Views

In the previous discussion, we assumed the simplest case where there exist advertised views that accomplish all the goals of the request. However, in many cases there may be no single advertisements that accomplish them all. This would require views from multiple advertisements to be combined in order to meet the requester's needs. In order to handle these cases, the described first and third phases need to be enhanced.

For example, in Phase 1, the search for alternatives must also search for combinations of advertisements that accomplish all goals. The merge process can be used again to verify that these combinations of advertisements are compatible with each other in addition to meeting the constraints of the request. For combined views belonging to a single provider, the combination (and its alternatives involving nonvital nodes) are grouped together as a single view.

The requester in Phase 3 must be aware that returned views do not necessarily accomplish all goals. Any E-Service bids that only satisfy some of the goals must be combined with other returned views to form complete solutions.

5. Implementation

In our previous work, we proposed to use mobile agents as a platform for establishing VE's [6]. Our goal is to implement our scheme described in the previous section on this platform. The idea is to use mobile agents to perform the phases of the scheme. The requester dispatches an agent with its request. The agent visits trading servers, and spawns copies of itself to deliver alternatives to different providers. It then gathers all returned service bids together and delivers the results back to the requester.

A core concept in our scheme for integrating E-Services is the merge process. It is this process that verifies whether or not different workflow views are compatible with each other. It is also responsible for adding nonvital nodes to views and verifying that a view is compatible with a business' existing workflow repository. The merge process can even be used to generate the outsourcing requests.

Repeat

- $Ad1 = Ad \in DB \mid < \text{selection conditions} >$
- $AdV = \text{projection}(Ad1, \{a \mid a \in Ad1 \wedge a.\text{significance} = \text{vital}\})$
- $RqV = \text{projection}(Rq, \{a \mid a \in Rq \wedge a.\text{significance} = \text{vital}\})$
- $BV = \text{bind}(AdV, RqV)$
- Repeat
 - $Nodes = \text{projection}(Ad1, \text{selected nonvital} \in Ad1) \cup \text{projection}(Rq, \text{selected nonvital} \in Rq)$
 - $Bx = \text{addNodes}(BV, Nodes)$
 - $\text{If } Bx \neq \text{null} \rightarrow A(P_i) = A(P_i) \cup Bx$
- Until all combinations of nonvitals found or termination criteria met

Until all Ads found that meet $< \text{selection conditions} >$ or termination criteria met

Figure 3. Service For VE Terms

Merging is not a trivial problem. It can be formulated as a *Constraint Satisfaction Problem* or CSP, with temporal features. The process must consider temporal constraints, resource usage, and causal links (preconditions and effects). There has been a great deal of research done on similar problems by the Artificial Intelligence community [14, 17, 20]. A number of formalizations have been developed for variations with more or less expressivity. The two that most closely match our problem are the Disjunctive Temporal Problem (DTP) and the Conditional Disjunctive Temporal Problem (CDTP).

For solving DTPs we have developed and implemented a new algorithm called Epilitis [16], along with algorithms that convert CDTPs to DTPs so that they may be solved by it as well. Epilitis builds on plan merging techniques used in a tool called PMA (Plan Management Agent) [18]. Epilitis integrates a number of techniques for pruning the search space, some of which are Conflict Directed Backjumping, Removal of Subsumed Variables, Semantic Branching, and no-good learning. Epilitis is currently the most efficient algorithm for solving such problems, as experimental results have shown that it is two orders of magnitude faster than the previous state-of-the-art solver, on synthetic benchmark problems.

In the prototype system we are currently developing, we will use Epilitis for the merging process at the WfMS and the trading servers. The representation that Epilitis expects is nearly identical to our enhanced model of workflows; the mapping between the two is trivial. Merging with Epilitis has all the properties discussed in Section 3. Any conflicting tasks are identified with explanation, and a minimal number

of constraints are added. Plans with disjunctive temporal constraints are supported (for added flexibility), and duplicate nodes can be identified and pruned/combined.

Epilitis does not support the notion of significance (vital vs. nonvital tasks). However these are implemented in the higher-level layer that performs the phases of our scheme. Only this layer is aware of the vital/nonvital distinction. (This is the cause of some of the complexity in the search for alternatives, as all the different combinations of nonvitals must be attempted separately.) This layer also serves to interface Epilitis with a relational DBMS using Microsoft Access and MySQL that will be used to implement the Workflow Repositories.

The current version of Epilitis is written in LISP, but using JLinker we have interfaced it to the rest of our prototype which is being developed in Java. The new version of Epilitis currently being developed will be in Java as well.

6. Conclusions

We are concerned with integrating E-Services for the establishment of a VE, where such services are represented with workflows. We have therefore created algorithms that make use of existing plan merging and temporal reasoning algorithms from the AI literature. Our scheme is sound, in that the workflows it returns as possible merge candidates are guaranteed to be correct. It is also complete, in that it will find all such candidates, given sufficient time. It further ensures that the merge candidates are compatible with all businesses involved in the VE. It can create the outsourcing requests based on identified conflicts, handle any number of

nodes and workflows to be outsourced, and is flexible in that it can build a VE using multiple providers, each with their own set of constraints. Our scheme also takes into consideration that workflows have both vital and nonvital steps, and appropriately considers them in its search.

In our proposed system, the merging process is built with existing AI algorithms. The specific algorithm, Epilitis, is the best algorithm available at this time. It has been implemented and is a fully functional and working plan merging tool. Currently we are developing our prototype system. Our goal is to evaluate its performance in terms of speed and memory usage. Another area we intend to explore is its use as a plan/workflow repair system that would replace broken or invalidated nodes or views with alternatives, possibly located in different databases on various machines.

Recently, there have been a variety of platforms developed with business to business E-services and Virtual Enterprises in mind. E*speak [3] from HP, VortexXML [4], and CrossFlow [11] are examples. These systems provide various features for managing and monitoring VE's, along with some standards for communication. Such systems could potentially be augmented or used conjunctively with our scheme for automated VE establishment. We will investigate such possibilities as part of our future work.

References

- [1] Alonso G., D. Agrawal, A. El Abbadi and C. Mohan. Functionalities and Limitations of Current Workflow Systems. *IEEE Expert*, 12(5), 1997.
- [2] Casati F., S. Ceri, B. Pernici and G. Pozzi. Workflow Evolution, *Data & Knowledge Engineering*, 24(3): 211-238, 1998.
- [3] Casati F. and M. Shan. Definition, Execution, Analysis, and Optimization of Composite E-Services. Bulletin of the Technical Committee on Data Engineering, 24(1):30-35, 2001.
- [4] Christophides V., R. Hull, A. Kumar, and J. Simeon. Workflow Mediation using VortexXML. Bulletin of the Technical Committee on Data Engineering, 24(1):41-46, 2001.
- [5] Chrysanthos P. K.. Guest Editor's Introduction to Special Issue on Workflow Systems. *Distributed Systems Engineering*, 3(4):211-212, 1996.
- [6] Chrysanthos P., T. Znati, S. Banerjee, S. Chang. Establishing Virtual Enterprises by means of Mobile Agents. *Research Issues in Data Engineering*, 1999.
- [7] Cichocki A. and M. Rusinkiewicz. Migrating Workflows. *Workflow Management Systems and Interoperability*, A.Dogac et al. (Eds)Springer Verlag, Series F, Vol 164, pp. 339-355, 1998.
- [8] Davulcu H., M. Kifer, L. R. Pokorny, C. R. Ramakrishnan, I. V. Ramakrishnan, and S. Dawson. Modeling and Analysis of Interactions in Virtual Enterprises. *Research Issues in Data Engineering*, 1999.
- [9] Georgakopoulos D., M. Hornick and A. Sheth. "An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure." *Distributed and Parallel Databases*, 3(2), 1995.
- [10] Georgakopoulos D., H. Sinha, K. Huff and B. Hurwitz. "Monitoring Multi-organizational Processes." *Proc. of the 11th Int'l Conf. on Parallel and Distributed Computing Systems*, pp. 75-80, 1998.
- [11] Grefen P., K. Aberer, H. Ludwig, and Y. Hoffner. CrossFlow: Cross-Organizational Workflow Management for Service Outsourcing in Dynamic Virtual Enterprises. Bulletin of the Technical Committee on Data Engineering, 24(1):53-58, 2001.
- [12] Han Y. and A. Sheth. On Adaptive Workflow Modeling. *Proc. of the 4th Int'l Conf. on Information Systems Analysis and Synthesis*, pp. 108-116, 1998
- [13] Jablonski S. et al. External and Internal Support Services in Workflow Management Systems. *Proc. of the 11th Int'l Conf. on Parallel and Distributed Computing Systems*, pp. 81-86, 1998.
- [14] V. Kumar. Algorithms for Constraint-Satisfaction Problems: A Survey. *AI Magazine*, 13(1):32-44, 1992.
- [15] Ramamritham K. and P. K. Chrysanthos. *Advances in Concurrency Control and Transaction Processing*, IEEE Computer Society Press, 1997.
- [16] Tsamardinos I. Constraint-Based Temporal Reasoning Algorithms with Applications to Planning. *Ph.D. Thesis. University of Pittsburgh Intelligent Systems Program*, 2001
- [17] Tsamardinos I., M. E. Pollack, et al. Merging Plans with Quantitative Temporal Constraints, Temporally Extended Actions, and Conditional Branches. *Artificial Intelligence Planning and Scheduling (AIPS'00)*, Breckenridge, Colorado, USA, 2000
- [18] Tsamardinos I., M.E. Pollack, et al. Adjustable Autonomy for a Plan Management Agent. *AAAI Spring Symposium on Adjustable Agents.*, 1999.
- [19] Workflow Management Coalition, Technology & Glossary, Document Number WFMC-TC-1011, June 1996.
- [20] Q. Yang. Intelligent Planning: A Decomposition and Abstraction Based Approach. Springer, 1997.

Flexible Dispatch of Disjunctive Plans

Ioannis Tsamardinos¹, Martha E. Pollack², and Philip Ganchev¹

¹ Intelligent Systems Program, University of Pittsburgh, Pittsburgh, PA 15260 USA
tsamard@eecs.umich.edu, ganchev@cs.pitt.edu

² Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48103 USA
pollackm@eecs.umich.edu

Abstract. Many systems are designed to perform both planning and execution: they include a plan deliberation component to produce plans that are then dispatched to an execution component, or *executive*, which is responsible for the performance of the actions in the plan. When the plans have temporal constraints, dispatch may be non-trivial, and the system may include a distinct *dispatcher*, which is responsible for ensuring that all temporal constraints are satisfied by the executive. Prior work on dispatch has focused on plans that can be expressed as Simple Temporal Problems (STPs). In this paper, we sketch a dispatch algorithm that is applicable to a much broader set of plans, namely those that can be cast as Disjunctive Temporal Problems (DTPs), and we identify four key properties of the algorithm.

1 Introduction

Many systems are designed to perform both planning and execution: they include a plan deliberation component to produce plans that are then dispatched to an execution component, or *executive*, which is responsible for the performance of the actions in the plan. When the plans have temporal constraints, dispatch may be non-trivial, and the system may include a distinct *dispatcher*, which is responsible for ensuring that all temporal constraints are satisfied by the executive. Prior work on plan dispatch [1-3] has focused on plans that can be represented as Simple Temporal Problems (STP) [4]. In this paper, we sketch a dispatch algorithm that is applicable to a much broader set of plans, those that can be cast as Disjunctive Temporal Problems (DTPs), and identify four key properties of the algorithm.

2 Disjunctive Temporal Problems

Definition. A *Disjunctive Temporal Problem (DTP)* is a constraint satisfaction problem $\langle V, C \rangle$, where V is a set of variables (or nodes) whose domains are the real numbers, and C is a set of disjunctive constraints of the form $C_i: l_i \leq x_i - y_i \leq u_i \vee$

$\dots \vee l_n \leq x_n - y_n \leq u_n$, such that for $1 \leq i \leq n$, x_i and y_i are both members of V , and l_i, u_i are real numbers. An **exact solution** to a DTP is an assignment to each variable in V satisfying all the constraints in C . If a DTP has at least one exact solution, it is **consistent**.

A DTP can be seen as encoding a collection of alternative Simple Temporal Problems (STPs). To see this, note that each constraint in a DTP is a disjunction of one or more STP-style inequalities. Let C_{ij} be the j -th disjunct of the i -th constraint of the DTP. If we select one disjunct C_{ij} from each constraint C_i , then the set of selected disjuncts forms an STP, which we will call a **component STP** of a given DTP. It is easy to see that a DTP D is consistent if and only if it contains at least one consistent component STP. Moreover, any solution to a consistent component STP of D is also clearly an exact solution to D itself.

Definition. A(n inexact) **solution** to a DTP is a consistent component STP of it. The **solution set** for a DTP is the set of all its solutions.

When we speak of a solution to a DTP, we shall mean an inexact solution. Plans can be cast as DTPs by including variables for the start and end points of each action.

3 A Dispatch Example

Consider a very simple example of a plan with three actions, P , Q , and R . (For presentational simplicity, we assume each action is instantaneous and thus represented by a single node). P must occur in the interval $[5,10]$ and Q in the interval $[15,20]$; P and Q must be separated by at least 6 time units; and R must be performed either the interval $[11,12]$ or $[21,22]$. The plan as described can be represented as the following DTP: $\{C1. 5 \leq P - TR \leq 10 \vee 15 \leq P - TR \leq 20; C2. 5 \leq Q - TR \leq 10 \vee 15 \leq Q - TR \leq 20; C3. 6 \leq P - Q \leq \infty \vee 6 \leq Q - P \leq \infty; C4. 11 \leq R - TR \leq 12 \vee 21 \leq R - TR \leq 22\}$. (Note that TR , the time reference point, denotes an arbitrary starting point.) This DTP has four (inexact) solutions: $\{STP_1: C_{11}, C_{22}, C_{32}, C_{41}; STP_2: C_{11}, C_{22}, C_{32}, C_{42}; STP_3: C_{12}, C_{21}, C_{31}, C_{41}; STP_4: C_{12}, C_{21}, C_{31}, C_{42}\}$.

Definition: An STP variable x is **enabled** if and only if all the events that are constrained to occur before it have already been executed. A DTP variable x is **enabled** if and only if it has a consistent component STP in which x is enabled.

In STP_1 , both P and R are initially enabled, while in STP_3 and STP_4 , Q is initially enabled. Hence, all three actions are initially enabled for the DTP. Enablement is a necessary but not sufficient condition for execution: an action must also be *live*, in the sense that the temporal constraints pertaining to its clock time of execution are satisfied. In the current example, none of the actions are initially live. The first action to become live is P , at time 5. An action is *live* during its *time window*.

Definition: The *time window of an STP variable* x is a pair $[l, u]$ such that $l \leq x - TR \leq u$, and for all l', u' such that $l' \leq x - TR \leq u'$, $l' \leq l$ and $u \leq u'$. Given a set of consistent component STPs for a DTP, we will write $TW(x, i)$ to denote the time window for variable x in the i^{th} such STP. The *upper bound* of a time window $[l, u]$ for x in STP i , written $U(x, i)$, is u . The *time window of a DTP variable* x is $TW(x) = \bigcup_{i \in S} TW(x, i)$, where S is the solution set of D .

The dispatcher can provide information about when actions are enabled and live in an *Execution Table (ET)*. This is a list of ordered pairs, one for each enabled action. The first element of the entry specifies the action, and the second is a list of the convex intervals in that element's time window. For our example, then, the initial ET would be: $\{ \langle P, \{[5, 10], [15, 20]\} \rangle, \langle Q, \{[5, 10], [15, 20]\} \rangle, \langle R, \{[11, 12], [21, 22]\} \rangle \}$. The ET summarizes the information in the solution STPs so that the executive does not have to handle them directly.

The ET provides information about what actions *may* be performed, but it does not provide enough information for the executive to determine what actions *must* be performed. To see this, note that the ET just given does not indicate that there is a problem with deferring both P and Q until after time 10. However, such a decision would lead to failure: if the clock time reaches 11 and neither P nor Q has been executed, then all four solutions to the DTP will have been eliminated. Thus, in addition to the information in the ET, the dispatcher must also provide a second type of information to the executive. The *deadline formula (DF)* provides the executive with information about the next deadline that must be met.

In the next section, we explain how to calculate the DF, which is more complicated than computing the ET. Here we simply complete the example, by illustrating how the ET and the DF would be updated as time passes. The initial DF would indicate that either P or Q must be executed by time 10. Suppose that at time 8, action P is executed. At this point, STP_3 and STP_4 are no longer solutions. The ET then becomes $\{ \langle Q, \{[15, 20]\} \rangle, \langle R, \{[11, 12], [21, 22]\} \rangle \}$ and the DF is trivially " Q by 20". In this case, an update to ET and DF resulted because an activity occurred. However, updates may also be required when an activity does not occur within an allowable time window. For example, if R has still not executed at time 13, then its entry in the ET should be updated to be just the singleton $[21, 22]$, with no changes required to the DF. The example presented in this section contains variables with very little interaction. In general, there can be significantly more interaction amongst the temporal constraints, and the DF can be arbitrarily complex.

4 The Dispatch Algorithm

We now sketch our algorithm for the dispatch of plans encoded as DTPs. The input is a DTP and the output is an Execution Table (ET) and a Deadline Formula (DF). For each pair $\langle x, TW(x) \rangle$ in ET, x must be executed some time within $TW(x)$. It is up to the executive to decide exactly when. The DF imposes the constraint that F has to

hold by time t , where a variable that appears in the DF becomes true when its corresponding event is executed.

The dispatch algorithm will be called in three circumstances: (1) when a new plan needs to have its dispatch information initialized, at or before time TR ; (2) when an event in the DTP is executed; (3) when an opportunity for execution passes because the clock time passes the upper bound of a convex interval in the time window for an action that has not yet been executed. Pseudo-code is provided in Figure 1. Space constraints preclude detailed description of the algorithm (but see [5]). Here we simply illustrate the procedure for computing the DF, the most interesting part of the algorithm.

Recall the example above. Initially, at time TR , the DTP has four solutions. To determine the initial DF, we consider the next critical moment, NC , which is the next time at which any action must be performed. This time is equal to the minimal value of all the upper bounds on time windows for actions, i.e., it is $\min\{U(x,i) \mid x \text{ is an action in the DTP, and } i \text{ is a solution STP}\}$. For instance, in our example DTP, $U(P, 1) = U(P, 2) = 10$. The actions that may need to be executed by NC are those x such that $U(x,i) = NC$ for some STP i . We create a list $UMIN$ containing ordered pairs $\langle x,i \rangle$ such that $U(x,i) = NC$. In our current example, $UMIN = \{\langle P, 1 \rangle, \langle P, 2 \rangle, \langle Q, 3 \rangle, \langle Q, 4 \rangle\}$. Now we perform the interesting part of the computation. If $\langle x,i \rangle$ is in $UMIN$, it means that unless x is executed by time NC , STP_i will cease to be a solution for the DTP. It is acceptable for STP_i to be eliminated from the solution set only if there is at least one alternative STP that is not simultaneously eliminated. This is exactly what the deadline formula ensures: that at the next critical moment, the entire set of solutions will not be simultaneously eliminated. We thus use a minimal set cover algorithm to compute all sets of pairs $\langle x,i \rangle$ in $UMIN$ such that the i values form a minimal cover of the set of solution STPs. In our example, there is only one minimal cover, namely the entire set $UMIN$. Thus, the initial DF specifies that P or Q must be executed by time 10: $\langle P \vee Q, 10 \rangle$. In general, there may be multiple minimal covers of the solution STPs: in that case, each cover specifies a disjunction of actions that must be performed by the next critical time. For instance, suppose that some DTP has four solution STPs, and that at time TR , $U(L, 1) = U(L, 2) = U(M, 3) = U(M, 4) = U(N, 4) = U(S, 3) = 10$. Then by time 10 either L or M must be executed; additionally, at least one of L or N or S must be executed. The corresponding DF is $\langle (L \vee M) \wedge (L \vee N \vee S), 10 \rangle$.

5 Formal Properties of the Algorithm

The role of a dispatcher is to notify the executive of when actions may be executed and when they must be executed. Informally, we will say that a dispatch algorithm is *correct* if, whenever the executive executes actions according to the dispatch notifications, the performance of those actions respects the temporal constraints of the underlying plan. Obviously, dispatch algorithms should be correct, but correctness is not enough. Dispatchers should also be *deadlock-free*: they should provide enough information so that the executive does not violate a constraint through inaction. A

Initial-Dispatch (DTP D)

1. Find all n solutions (consistent component STPs) to D , calculate their distance graphs, and store them in Solutions $[i]$. Associate each solution with its (integer-valued) index.
2. Set the variable TR to have the status Executed, and assign $TR=0$.
3. Compute-Dispatch-Info(Solutions).

Update-for-Executed-Event (STP $[i]$ Solutions)

1. Let x be the event that was just executed, at time t .
2. Remove from Solutions all STPs i for which $t \notin TW(x, i)$.
3. Propagate the constraint $t \leq x - TR \leq t$ in all remaining Solutions.
4. Mark x as Executed.
5. Compute-Dispatch-Info (Solutions).

Update-for-Violated-Bounds (STP $[i]$ Solutions)

1. Let $U = \{U(x, k) \mid U(x, k) < \text{Current-Time}\}$
2. Remove from Solutions all STPs k that appear in U .
3. Compute-Dispatch-Info (Solutions).

Compute-Dispatch-Info (STP $[i]$ Solutions)

1. For each event x in Solutions
2. {If x is enabled
3. $ET = ET \cup \langle x, TW(x) \rangle$.
4. Let U = the set of upper bounds on time windows, $U(x, i)$ for each still unexecuted action x and each STP i .
5. Let NC , the next critical time point, be the value of the minimum upper bound in U .
6. Let $U_{MIN} = \{U(x, i) \mid U(x, i) = NC\}$.
7. For each x such that $U(x, i) \in U_{MIN}$, let $S_x = \{i \mid U(x, i) \in U_{MIN}\}$
8. {Initialize $F = \text{true}$;
9. For each minimal solution $MinCover$ of the set-cover problem (Solutions, $\cup S_x$), let $F = F \wedge (\vee_{x \mid S_x \in MinCover} x)$.
10. $DF = \langle F, NC \rangle$.

Figure 1. The Dispatch Algorithm

third desirable property for dispatchers is **maximal flexibility**: they should not issue a notification that unnecessarily eliminates a possible execution, i.e., an execution that respects the constraints of the underlying plan. Finally, we will require dispatch algorithms to be **useful**, in the sense that they really do some work. Usefulness will be defined as producing outputs that require only polynomial-time reasoning on the part of the executive. Without a requirement of usefulness, one could achieve the other three properties by designing a DTP dispatcher that simply passed the DTP representation of a plan on to the executive, letting it do all the reasoning about when to execute actions.

Our dispatch algorithm has these four properties, as proved in [5]. The proofs depend on a more precise notion of how the dispatcher and the executive interact. The dispatcher issues a *notification sequence*, a list of pairs $\langle ET, DF \rangle_1 \dots \langle ET, DF \rangle_n$, with a new notification issued every time an event is executed or an upper bound is passed. The executive performs an *execution sequence*, a list $x_1 = t_1, \dots, x_n = t_n$ indicating that event x_i is executed at time t_i , subject to the restriction that $j > i \Rightarrow t_j > t_i$. An execution sequence is complete if it includes an assignment for each event in the original DTP; otherwise it is partial. The notification and execution sequences will be interleaved in an *event sequence*. We associate each execution event with the preceding notification, writing $\text{Notif}(x_i)$ to denote the notification of event x_i .

Definition. An execution sequence E respects a notification sequence N iff

1. For each execution event $x_i = t_i$ in E , $\langle x_i, TW(x_i) \rangle$ appears in ET of $\text{Notif}(x_i)$ and $t_i \in TW(x_i)$, i.e., each event is performed in its allowable time window.
2. For each $DF = \langle F, t \rangle$ in N , $\{x_i \mid x_i = t_i \in E \text{ and } t_i \leq t\}$ satisfies F . That is, the execution sequence satisfies all the deadline formulae.

Theorem 1: The dispatch algorithm in Fig. 1 is correct, i.e., any complete execution sequence that respects its notifications also satisfies the constraints of D .

Theorem 2: The dispatch algorithm in Fig. 1 is deadlock-free, i.e., any partial execution that respects its notifications can be extended to a complete execution that satisfies the constraints of D .

Theorem 3: The dispatch algorithm in Fig. 1 is maximally flexible, i.e., every complete execution sequence that respects the constraints in D will be part of some complete event sequence.

Theorem 4: The dispatch algorithm in Fig. 1 is useful, i.e., generating an execution sequence is polynomial in the size of the notifications.

References

1. Muscettola, N., P. Morris, and I. Tsamardinos. Reformulating Temporal Plans for Efficient Execution. in Proceedings of the 6th Conference on Principles of Knowledge Representation and Reasoning. 1998.
2. Tsamardinos, I., P. Morris, and N. Muscettola, Fast Transformation of Temporal Plans for Efficient Execution, in Proceedings of the 15th National Conference on Artificial Intelligence. 1988, AAAI Press/MIT Press: Menlo Park, CA. p. 254-261.
3. Wallace, R.J. and E.C. Freuder, Dispatchable Execution of Schedules Involving Consumable Resources, in Proceedings of the 5th International Conference on Artificial Intelligence Planning and Scheduling. 2000.
4. Dechter, R., I. Meiri, and J. Pearl, Temporal Constraint Networks. Artificial Intelligence, 1991. 49: p. 61-95.
5. Tsamardinos, I., Constraint-Based Temporal Reasoning Algorithms, with Applications to Planning. 2001.

Flexibility Measures of Sets of Plans

Philip Ganchev

philip@cs.pitt.edu

Intelligent Systems Program

University of Pittsburgh

2001.06.10.

1. Motivation

Many systems are designed to perform both planning and execution: they include a plan generation component to produce plans that are then dispatched to an execution component, or *executive*, which is responsible for the performance of the actions in the plan. When the plans have temporal constraints, dispatch may be non-trivial, and the system may include a distinct *dispatcher*, which is responsible for ensuring that all temporal constraints are satisfied by the executive. Such a dispatcher is the Dispatch Algorithm, [Tsamardinos, Pollack, Ganchev, 2001], which I will refer to as the TPG Dispatch Algorithm, or TPG, for clarity. In this project, I suggest a modification of the algorithm to increase its efficiency; in service of this, I also introduce a useful novel concept: flexibility of a set of plans.

The nature and difficulty of the dispatch problem depends upon the temporal representation used in the plans. When all actions have fixed times, plan dispatch is straightforward: the dispatcher just submits the fixed schedule to the executive, which then has to perform the actions according to that schedule. However, to allow for the possibility of unanticipated events, it is often desirable to use plans with looser temporal constraints. Much work has been done on plans represented as instances of the *Simple Temporal Problem* (STP).

Definition 1. A *Simple Temporal Problem* (STP) is a constraint satisfaction problem $\langle V, C \rangle$, where

- V is a set of *variables* (or: *nodes*) representing events, and their domains are the real numbers, and
- C is a set of *constraints* of the form $C_i: l \leq x - y \leq u$, such that $x, y \in V$, and l, u are real numbers.

The TPG Dispatcher applies to a broader set of plans: ones that can be represented as *Disjunctive Temporal Problems* (DTPs). The DTP formalism is strictly more expressive than the STP.

Definition 2. A *Disjunctive Temporal Problem (DTP)* is a constraint satisfaction problem $\langle V, C \rangle$, where

- V is a set of variables (or nodes) representing events, and their domains are the real numbers, and
- C is a set of disjunctive constraints of the form $C_i: l_i \leq x_1 - y_1 \leq u_1 \vee \dots \vee l_n \leq x_n - y_n \leq u_n$, such that for $1 \leq i \leq n$, $x_i, y_i \in V$, and l_i, u_i are real numbers.

Definition 3. An *exact solution* to a DTP is an assignment to each variable in V such that all the constraints in C are satisfied. If a DTP has at least one exact solution, it is *consistent*.

Definition 4. A(n inexact) *solution* to a DTP is a consistent component STP of it. The *solution set* for a DTP is the set of all its solutions.

The TPG Dispatcher has four required properties: it is correct, deadlock-free, maximally-flexible and useful. However, the algorithm relies upon access to the complete set of solutions to the given DTP, which it calculates in the first step, and stores in the list *Solutions*. For a DTP with m constraints, each of which has l disjuncts, there are in the worst case $\Theta(l^m)$ solutions. Each one takes polynomial time to find in the worst case, so the first step of the algorithm would take exponential time in the size of the DTP.

2. Incremental Generation of Solutions

One way to avoid generating all the DTP solutions at once is to generate a subset of them and continue with the algorithm treating the subset as the complete solution set. However, any substantial decrease in the size of *Solutions* corresponds to a decrease in the ways of executing the plan. In particular, the property of Maximal Flexibility of the dispatcher is sacrificed, since any removed solution will never be part of a complete event sequence – the dispatcher’s notifications

instruct the executive against executing this STP. Since the TPG Dispatcher removes STPs from *Solutions* when they are no longer consistent with the current execution sequence, most of the STPs may soon be removed from the working solution set, forcing the dispatcher to commit to the few that remain, and making the plan brittle.

Another possibility is to generate a subset of solutions initially, and search for additional ones incrementally, as STPs are removed from *Solutions*. Again, the resulting dispatcher is not maximally flexible, for the same reason as above. However, it considers many more solutions than the method above.

To define a criterion of how many additional solutions the algorithm should search for (i.e. when to stop searching), we introduce the notion of the *flexibility* of a set of plans. The *notion of flexibility* roughly represents the freedom allowed in the execution of the set of plans. An exact definition of a *measure of flexibility* is suggested in the following section. Until then, we can think of the number of ways of assigning values to the variables, as an example of a measure of flexibility. Let us assume that we have defined such a measure as the function

$$f: \{X \mid X \text{ is a set of STPs}\} \rightarrow \mathbb{N}.$$

Whenever some STPs are removed from *Solutions*, the dispatcher checks whether $f(\text{Solutions})$ exceeds some flexibility threshold FT . If so, it continues like the TPG Dispatcher, otherwise it searches for more solutions until *Solutions* again reaches the desired level of flexibility. The pseudocode for the algorithm is shown in Figure 1 below.

Figure 1: Restore-Flexibility

Global boolean *exist-more-solutions*

Restore-Flexibility(STP-set *Solutions*, DTP *D*, Flexibility Threshold *FT*, Search time limit *l*)

1. *time-used* \leftarrow 0
2. **while** $f(\text{Solutions}) < FT$ **and** *time-used* $< l$ **and** *exist-more-solutions* **do**
3. *STP* \leftarrow *Search-For-New-Solution*(*D*, $l - \text{time-used}$, *exist-more-solutions*)

4. **if** *exist-more-solutions* **then**
5. add *STP* to *Solutions*
6. **return** *Solutions*

Then, the subroutines of the dispatcher [1] become as shown in Figures 2 through 4 below.

Figure 2: Initial-Dispatch

Initial-Dispatch (DTP *D*)

6. *exist-more-solutions* \leftarrow **True**.
7. *Solutions* \leftarrow *Restore-Flexibility*(*Solutions*, *D*, *FT*, *I*).
Associate each solution with its (integer-valued) index.
8. Set the variable *TR* to have status **Executed**, and assign *TR* \leftarrow 0.
9. *Output-ET*(*Solutions*).
10. *Output-DF*(*Solutions*).

Figure 4: Update-for-Executed-Event

Update-for-Executed-Event (*Solutions*)

1. Let *x* be the event that was executed at time *t*.
2. Remove from *Solutions* all STPs *i* for which $t \notin \text{TW}(x, i)$.
3. Propagate the constraint $t \leq x - \text{TR} \leq t$ in all remaining STPs in *Solutions*.
4. *Solutions* \leftarrow *Restore-Flexibility*(*Solutions*, *D*, *FT*, *I*).
5. Mark *x* as **Executed**.
6. *Output-ET* (*Solutions*).
7. *Output-DF* (*Solutions*).

3. “Number-of-Executions” Flexibility Measure

One challenge in defining a measure of flexibility is that it can mean different things in different contexts. Another challenge is that it must be computed efficiently. Here I explore one definition of flexibility of a set of STNs.

For simplicity, let us first consider a single STP. If the domains of all the variables are discrete, we can define the flexibility in the way that is probably most intuitive:

Definition 5. The *number-of-executions flexibility of an STP with discrete variables* is the number of exact solutions to the STP, i.e. the number of assignments of values to the variables, consistent with the constraints.

Recall the definition of time window:

Definition. The *time window of an STP variable x* is a pair $[l, u]$ such that $l \leq x - TR \leq u$ satisfies all STP constraints, and for all l', u' such that $l' \leq x - TR \leq u'$ satisfies all STP constraints, $l' \leq l$ and $u \leq u'$. When every variable of an STP has a time window with finite bounds l and u , we will say the STP is *bounded*.

In many practical applications, the STPs can be bounded. Then, for the STNs where all the variables are discrete, the number of possible executions is the number of lattice points in a polytope P in \mathbf{R}^n :

$$|P \cap \mathbf{Z}^n|.$$

(\mathbf{Z} is the set of integers.)

To see this, consider an STP $S = \langle E, C \rangle$, with variables E , and constraints C . Suppose there are n variables x_i , each with domain the natural numbers \mathbf{N} . The space of all assignments to $\langle x_1, \dots, x_n \rangle$, is \mathbf{N}^n , where each variable corresponds to a dimension in \mathbf{N}^n . Each inequality constraint $C_i \in C$,

$$x_{i1} - x_{i2} \leq b_i$$

represents a half-space of \mathbf{R}^n on a plane with slope 1, -1 and 0 w.r.t. x_{i1} , x_{i2} and all other variables respectively. Any assignment satisfying C_i lies inside the half-space. Thus, if S is consistent, the intersection of all m half-spaces is nonempty, and the set of all STN constraints, C , represents a polyhedron P in \mathbf{R}^n . If every variable has an upper and a lower bound with respect to some point, then the polyhedron is bounded, i.e. a *polytope*. Combining this with the requirement for integral assignments to the variables, we get that the number of executions of the STP is $|P \cap \mathbf{Z}^n|$.

A summary of deterministic algorithms for computing the number of lattice points in polyhedra can be found in [Batvonik & Pommersheim, 1999]. Below, I focus on volume computation, since continuous variable STPs are more common.

For STPs where all the variables have continuous domains, the counterpart definition of flexibility is:

Definition 6. The *number-of-executions flexibility of an STP (with continuous variables)* is the size of the space of assignments of values to the variables, consistent with the constraints.

By the reasoning for the discrete case, this space is a volume in \mathbf{R}^n if the STP is consistent. Then the space of all consistent assignments (the flexibility) is the volume of the polytope P ,

$$\text{volume}(P).$$

Again, the STP must be bounded. For non-bounded STPs, the space of all consistent assignments is infinite, and we need another definition of flexibility.

We considered the case of a single STP. For a set of STPs, we must find the volume (or lattice points) of the union of the respective polytopes.

It is most accurate to use the volume for continuous STPs and the number of lattice points for discrete STPs. In fact, the number of lattice points can be used for STPs with continuous variables, by considering each variable to be discrete with certain *granularity*. Conversely, the volume can be used as an alternative measure for discrete STPs. However, there is no advantage to either of these, as the volume and the number of lattice points are equally costly to compute.

Exact computation of the volume of a polytope (or its number of lattice points) is #P-complete, and therefore NP-Hard. However, there are polynomial-time algorithms for approximating $\text{volume}(P)$ [Bollobás, 1997]. The best ones use rapidly mixing random walks and are *Fully-Polynomial Approximation Schemes* (FPRAS) based on [Dyer, Frieze, Kannan, 1991]. A FPRAS is a randomized algorithm that runs in time polynomial in the representation of P , $1/\varepsilon$ and $\log(1/\eta)$, and with probability at least $1-\eta$, produces an ε -approximation to $\text{volume}(P)$, for arbitrary $0 < \eta < 1$ and $0 < \varepsilon < 1$. An ε -approximation to $\text{volume}(P)$ is a number $\text{volume}\sim(P)$, s.t. $(1-\varepsilon) \text{volume}\sim(P) < \text{volume}(P) < (1+\varepsilon) \text{volume}\sim(P)$. The asymptotically most efficient algorithm to date is that of [Kannan, Lovász and Simonovitz, 1997], and is $O^*(n^5)$ ¹.

Unfortunately, there are no implementations of any approximate algorithms. There are several implementations of exact algorithms. The latest one is *Vinci*, [Bueler, Enge, Fukuda, 1998].

4. Timing Experiment

In order to assess the viability of this measure of flexibility, I timed *Vinci* on a range of randomly generated STPs.

4.1. Method

The program *Vinci* was run on a Micron Millennia personal computer with a Pentium III 700 MHz processor and 260 MB of RAM, running Windows NT 4.0. The experiments were done using a Lisp program *gen-test*. It generates a specified range of STPs by invoking *Gen-STN*, then invokes

¹ O^* is the “soft-O” notation usual for describing the speed of an FPRAS. It omits powers of $\log(n)$ and polynomials of $1/\varepsilon$ and $\log(1/\eta)$.

Vinci on each STP and summarizes the output. *Gen-STN* can generate a single STP in the format required by *Vinci*. It is an extension of *random-gen*, which is written by Ioannis Tsamardinos.

The STPs generated varied by several features: *nnodes*, *nedges*, *horizon* and *bound-param*. They are described below:

- ***nnodes*:** The number of STP variables, i.e. nodes. This includes *TR*, so $nnodes = 1 + \text{dimension}(P)$.
- ***nedges*:** The number of STP constraints.
- ***horizon*:** The upper bound on the uniform distribution from which the initial schedule is drawn; also the default constraints between each variable and *TR* have bounds $\pm horizon$.
- ***bound-param*:** The distribution from which the intervals are drawn that are combined with the initial schedule to produce the bounds for the constraints.

To better understand the purpose of the last two features, consider a summary of *gen-stn.lisp*'s algorithm, shown in Figure 5. Each STP has a corresponding distance graph, whose nodes correspond to variables and whose directed edges correspond to constraints between pairs of variables.

Figure 5: Gen-STN

Gen-STN (integers *nnodes*, *nedges*, *horizon*, *bound-param*)

1. Generate a random schedule for the variables. Assign each variable a value drawn from $\text{Uniform}[0, horizon]$.
2. Output the $2*nnodes$ default constraints $\pm x_i - \pm TR \leq horizon$.

There remain $nedges - 2*nnodes$ constraints to output, each of the form $x_{i1} - x_{i2} \leq b_i$, between distinct pairs of unique non-*TR* variables x_{i1}, x_{i2} .

3. Select the pairs x_{i1}, x_{i2} uniformly at random.

4. For each pair,

Choose a number r_i draw from $\text{Uniform}[0, \text{bound-param} - 1]$

Output the constraint $x_{i1} - x_{i2} \leq r_i + (\text{schedule-time}[x_{i1}] - \text{schedule-time}[x_{i2}])$.

We may expect that increasing the features will have the following effects respectively:

- ***nnodes***: Increases **volume(P)** since the number of dimensions increases;
Increases computation **time** and **space** because of the greater volume;
- ***nedges***: Decreases **volume(P)** (but less than *nnodes* increases it), because it makes the polytope more constrained;
Increases **time** and **space**, because additional constraint processing is required.
Increases **$P(\text{volume}(P)=0)$** because there are more trials to select a pair of nodes for both an upper and a lower bound.
- ***horizon***: Increases the expected bounds, hence **volume(P)**, **space** and **time**;
Increases the **variability of the bounds** because the distances in the schedule are drawn from a larger distribution.
- ***bound-param***: Increases the **variability of the bounds** because the r_i s are drawn from a larger distribution, hence
Decreases **$P(\text{volume}(P)=0)$** .

The significance of $P(\text{volume}(P)=0)$ will become clear when we discuss the results in Section 6.

We fix *horizon*, *bound-param* and *nnodes*, and vary *nedges* from the minimum to the maximum.

We test each configuration with 10 randomly generated STPs. We do this with *nnodes* = 6, ..., 11.

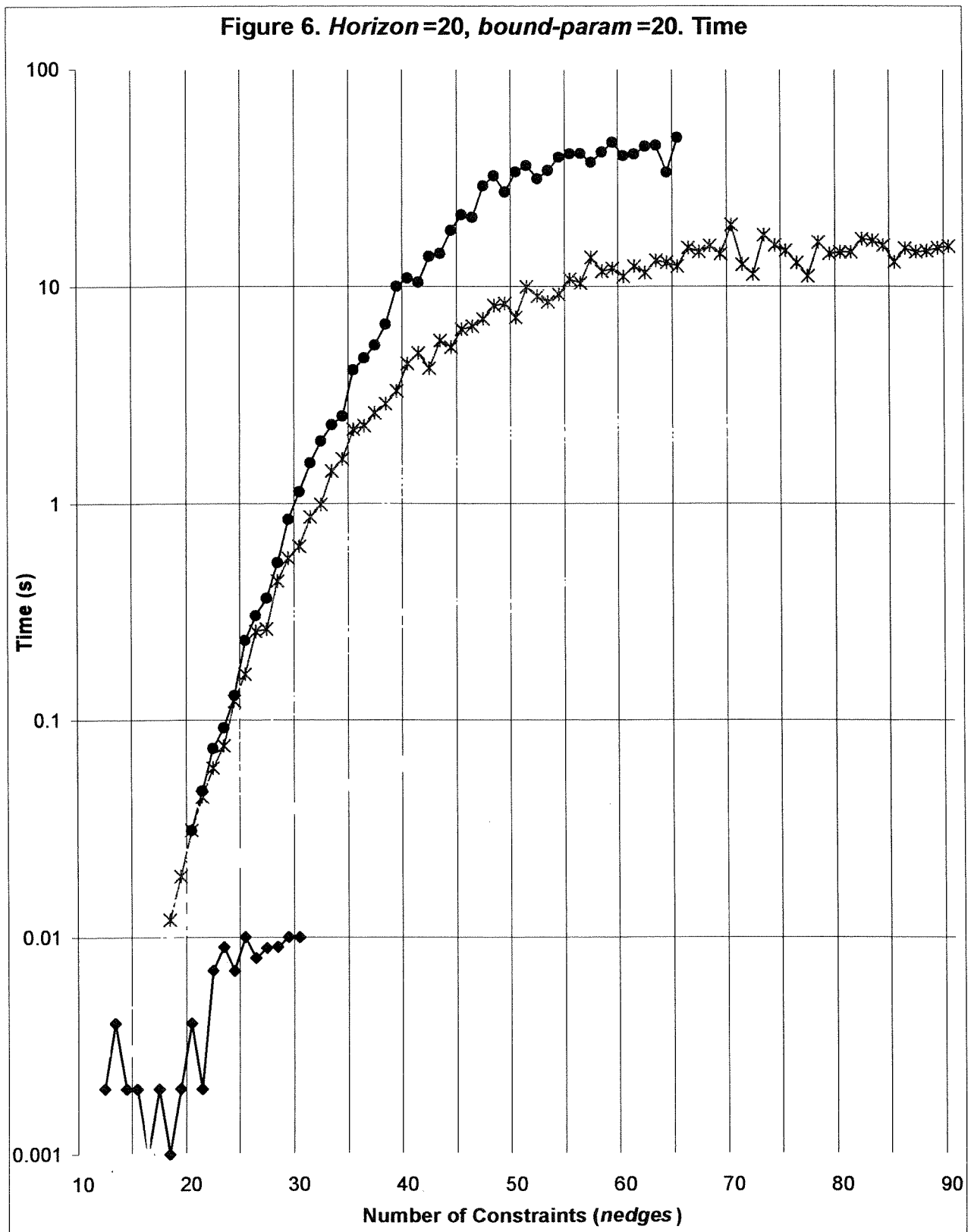
Then we repeat for other value-combinations of *horizon* and *bound-param*; the values used are (20, 20), (50, 20), (50, 50).

4.2. Results

The results from the experiments, the volume of the randomly generated STPs, time, space and volume, are summarized in the graphs in Figures 6 through 14 below. Each graph plots one of time, memory or volume against *nedges*, for *nnodes* = 6, ..., 11. The values shown are the arithmetic means of the runs of *Vinci* on the 10 randomly generated STPs. The graphs use a logarithmic scale on the vertical axis.

The graphs show the following results. Computation time and memory increase approximately exponentially with *nnodes* (the number of STP variables), and approximately linearly with *nedges* (the number of STP constraints). Volume (number-of-executions flexibility) decreases exponentially with the number of constraints, and increases approximately exponentially with the number of variables.

Legend					
—◆— <i>nnodes</i> =6	<i>nnodes</i> =7	<i>nnodes</i> =8	<i>nnodes</i> =9	—✱— <i>nnodes</i> =10	—●— <i>nnodes</i> =11



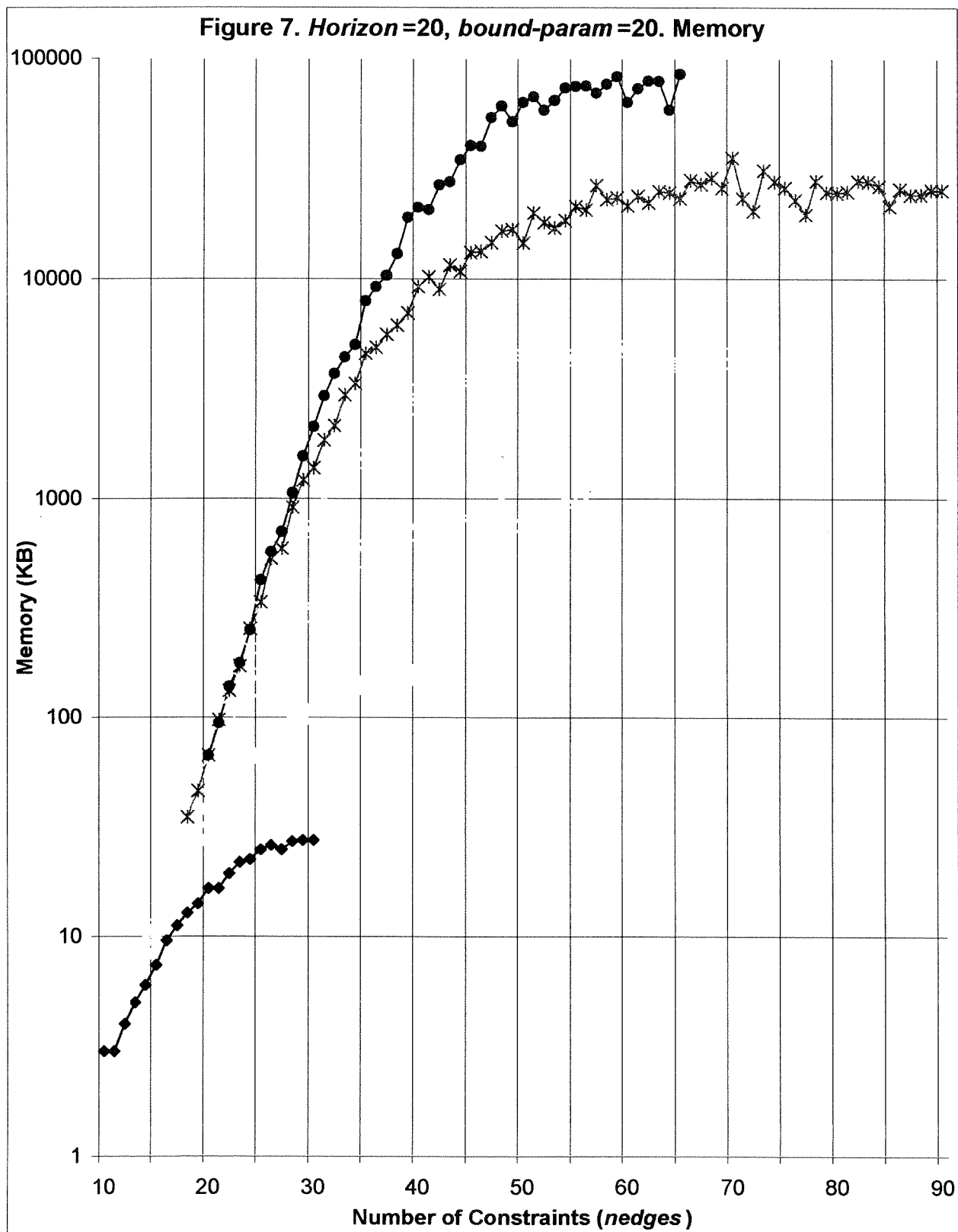
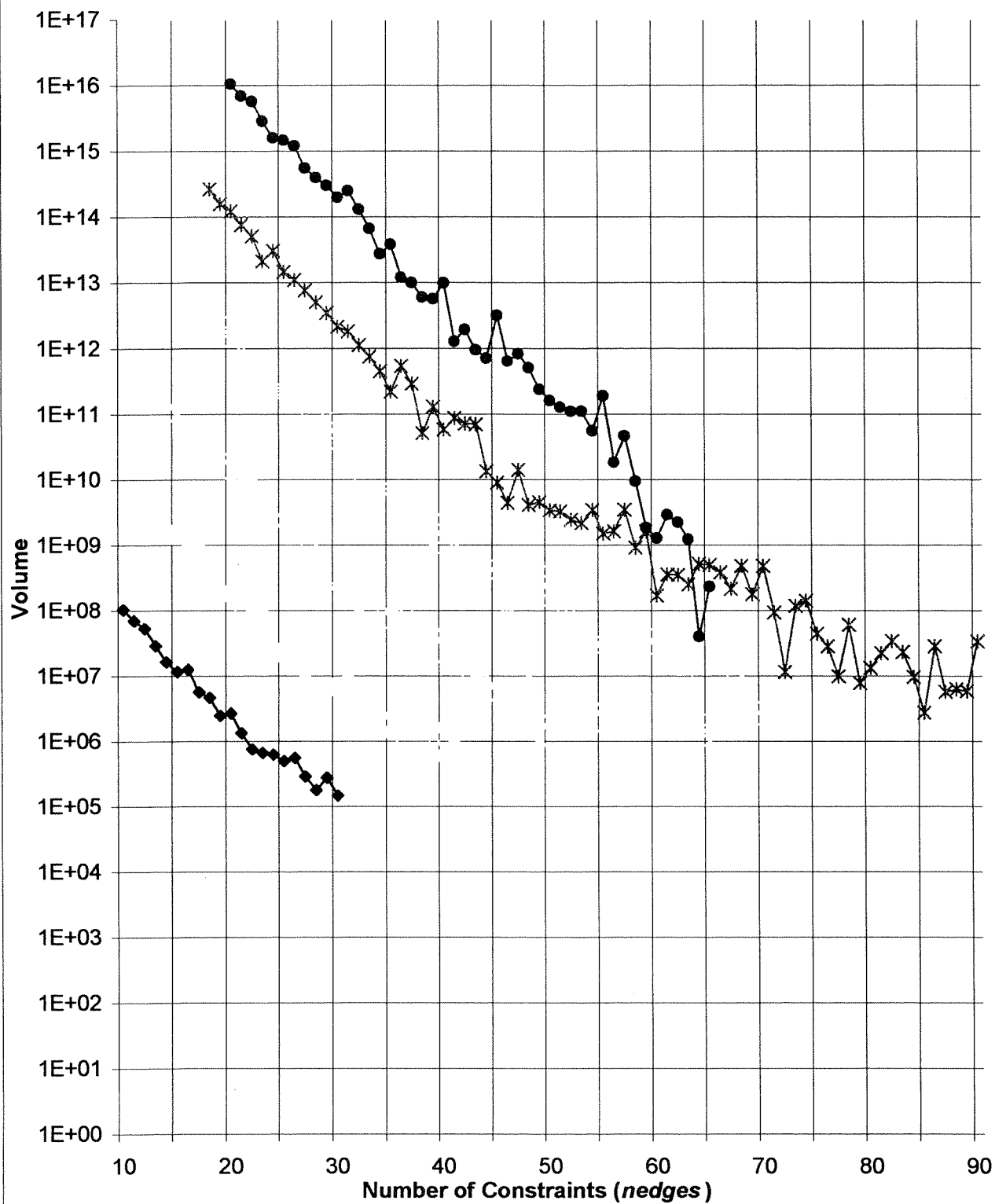
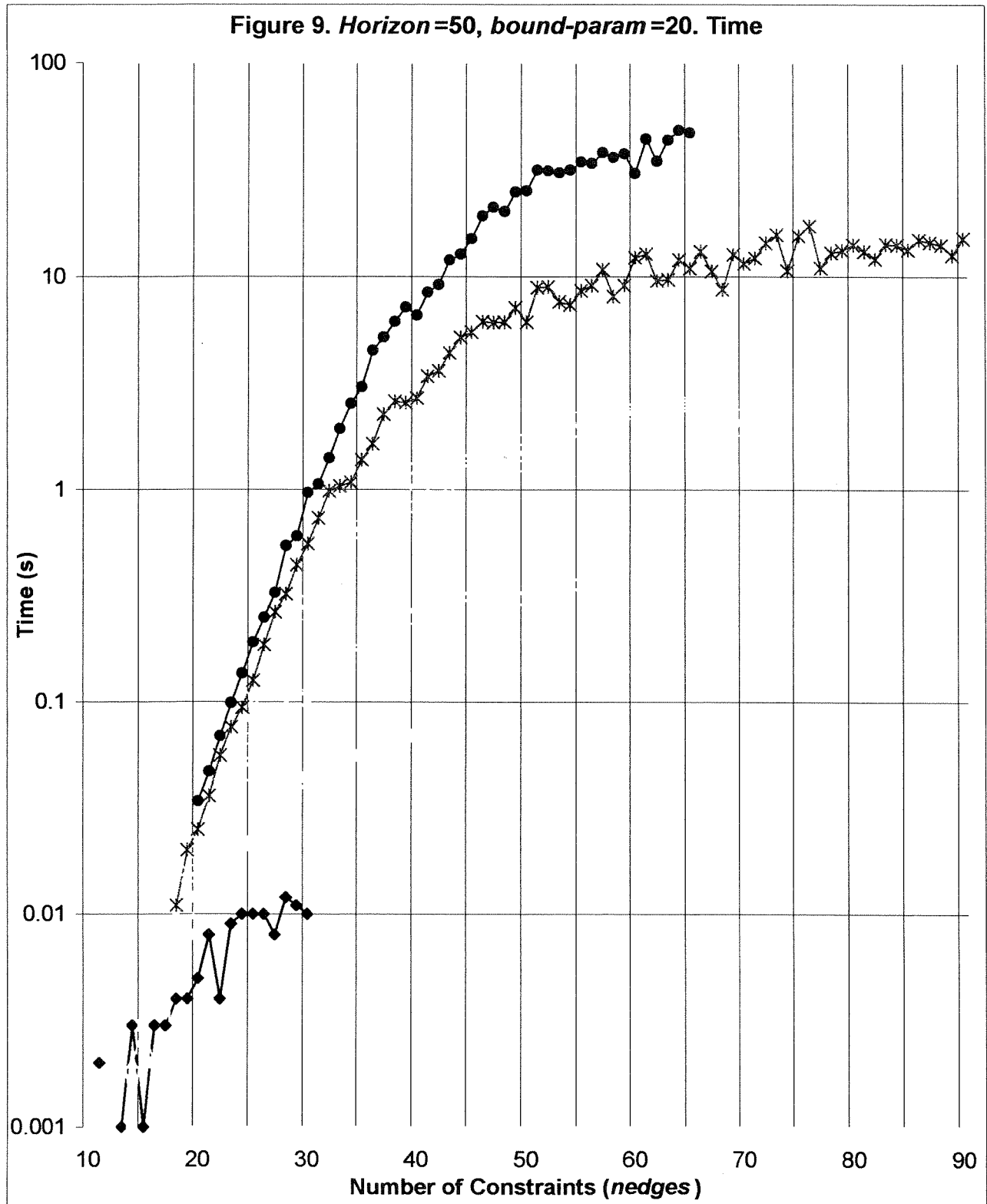
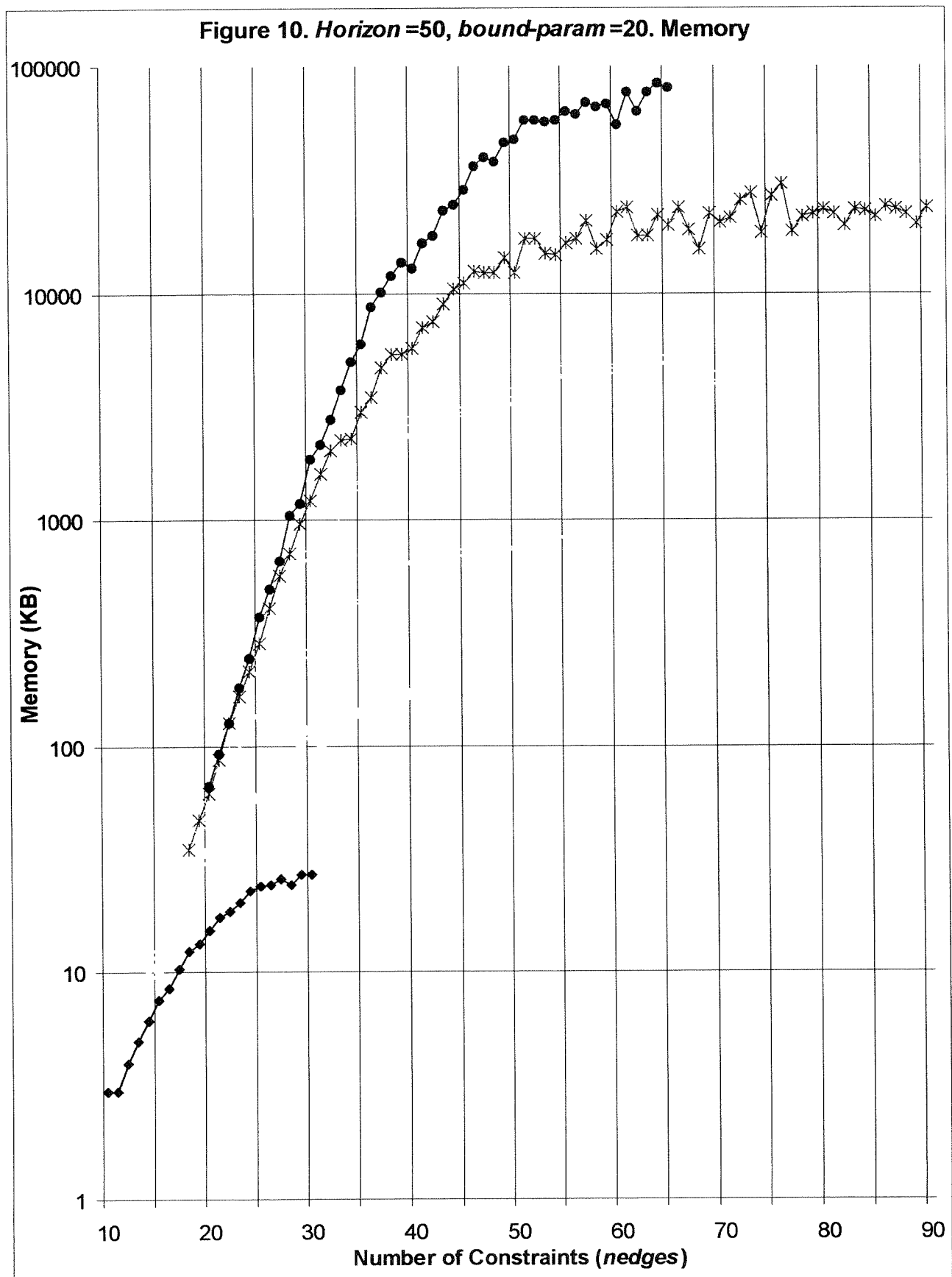
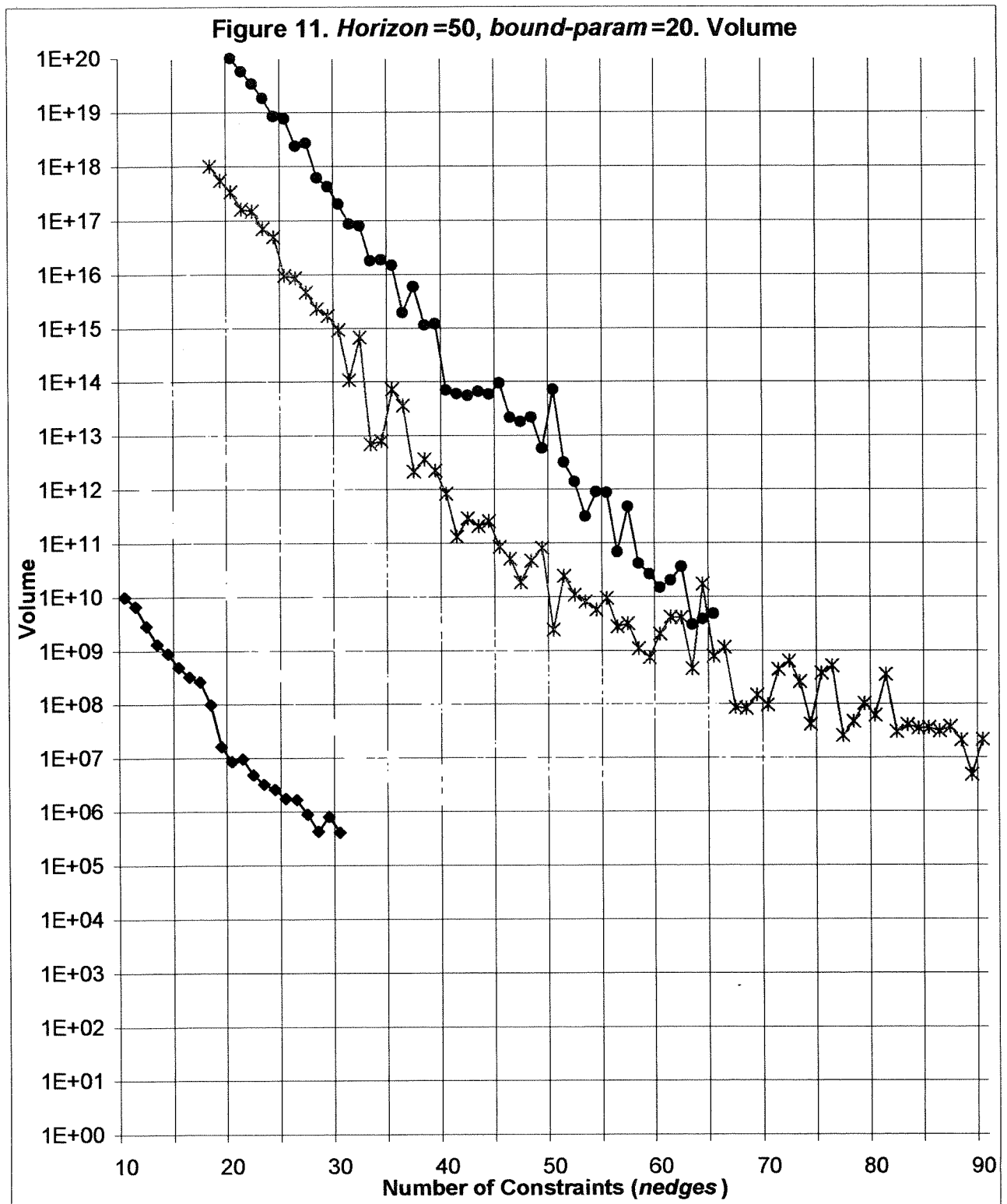


Figure 8. *Horizon=20, bound-param=20. Volume*









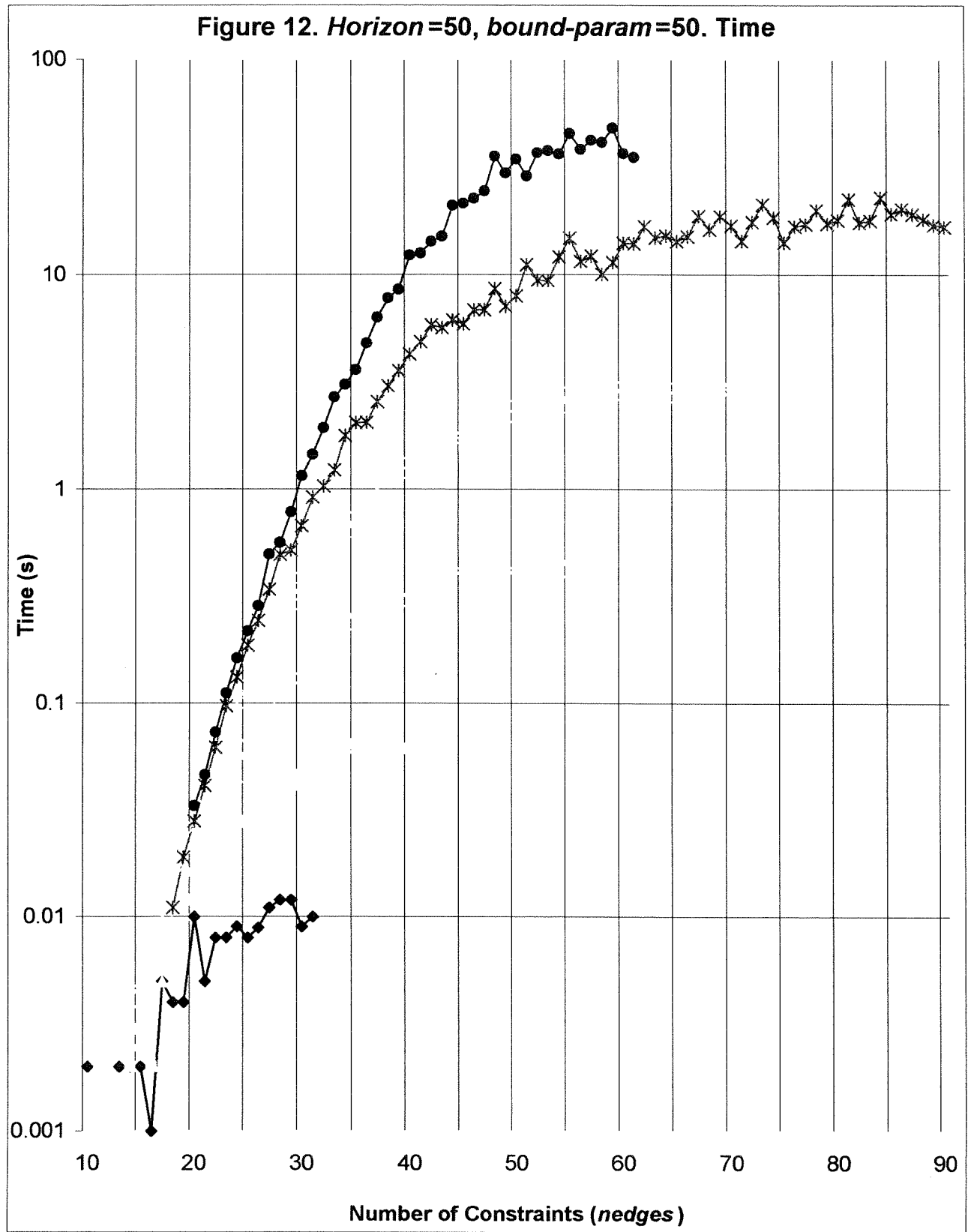
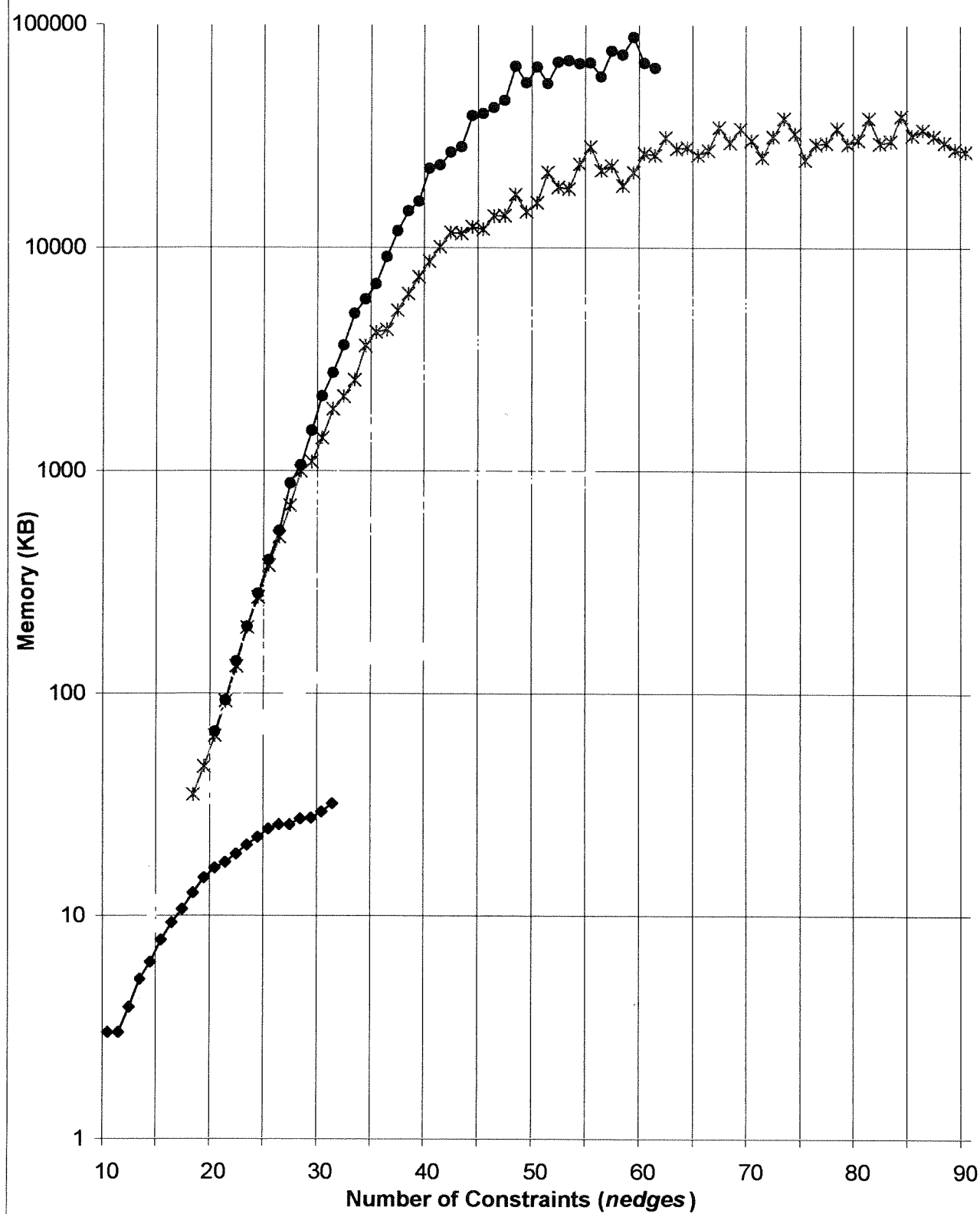
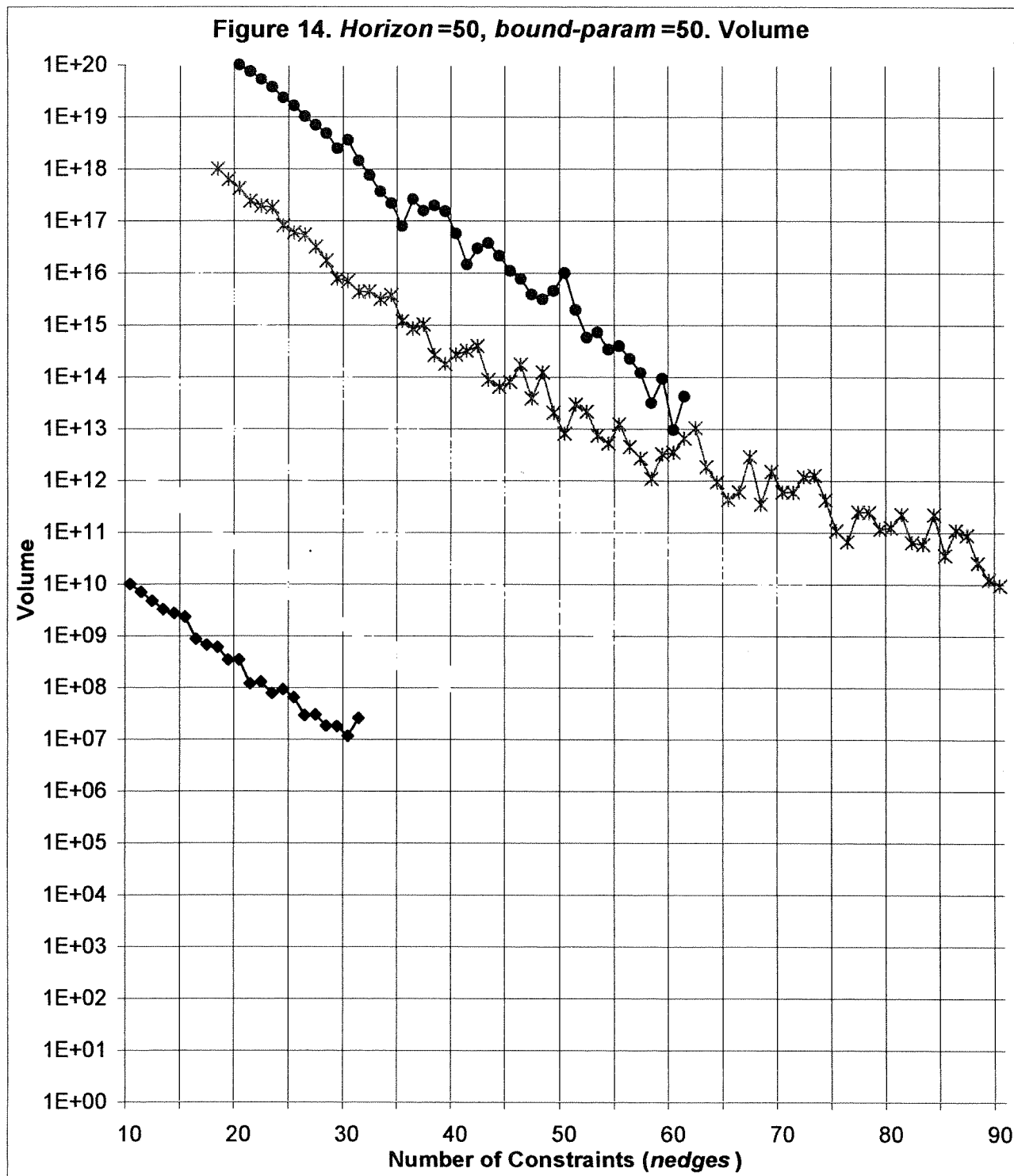


Figure 13. *Horizon=50, bound-param=50. Memory*





5. Discussion

The exponential growth of computation time and space means that exact computation of volume will be impractical as a flexibility measure for a DTP dispatcher with incremental generation of solutions. I suspect that it is as costly as initially generating all the solutions, as in the original TPG dispatcher; however, we have not done an empirical study of the latter.

Furthermore, many STPs with large *nnodes* (>10) and *nedges* made the program run out of memory and crash. *Vinci* can be run with the *storage level* option, *-sn*, to use less memory at the expense of computation time. Here *n* is a number; when the storage option is not used, *n* defaults to 20; lower *n* uses less memory.

I used the storage option to see the effect on computation time. First, I ran randomly generated STPs with *nnodes*=12 and increasing *nedges*, until a sequence of several STPs where *Vinci* ran out of memory. This happened at *nnodes*=45. Then I decreased the storage parameter until *Vinci* again had enough memory to complete the computations, and so on. The lowest storage level I tried was 4. The table in Figure 15 below summarizes the results.

Figure 15: The *-s* Option
***nnodes*=12, *horizon*=20, *bound-param*=20**

<i>Nedges</i> Range	Storage Level	Time Range (s)	Memory Range (Mb)
22 – 44	20 (default)	0.1 – 47	0.132 – 79.8
44 – 46	4	590 – 2197	5.2 – 8.8
46 – 50	6	39 – 207	29.5 – 87.8
50 – 59	5	185 – 1639	18.5 – 87.3

The computation times are again too large to be practical.

Another possibility, which I did not explore, is to compile the polytopes (the STPs) from a constraints representation (“H format”) into a vertices representation (“V format”) and use both representations in computing the volume; this would reduce the computation space, at the expense of the time for computing the V format. However, this is unlikely to sufficiently improve the speed compared to using a lower storage level.

In the experiment, some of the polytopes had volume 0. This occurred with large *nedges* and small *bound-param*. A 0 volume in *nnodes* dimensions occurs when one or more of the variables are constrained to a point. As an example, consider an STP with variables $\{TR, x, y\}$ and constraints $\{x-TR \leq 1, TR-x \leq 1, y-TR \leq 1, TR-y \leq -1\}$. The last two constraints imply $y=1$. So the solution space is a line segment $-1 \leq x \leq 1, y=1$. A line segment has volume 0 in 2D. However, there is more than one legal assignment to (x, y) . For instance, three are: $(-1,1), (0,1), (1,1)$. Thus, 0 volume does not correctly represent the number of executions in 2D, but it does in 1D. This is in effect the problem of computing the flexibility of a mixture of variables: here x acts as a continuous variable, y as a discrete one. This problem remains when we use an approximate algorithm instead of an exact one, since the algorithm would have to sample the space at a point in the correct hyperplane (in the example, the hyperplane $y=1$) and the likelihood of that is small.

There are at least two ways to correct the flexibility measure to account for this. The first way is to find all k variables constrained to a point, then compute the volume in $(nnodes-k)-D$. Unfortunately it is not sufficient to parse the constraints and find all pairs of variables constrained by the same bound from above and below, as in the example above. A 0 volume can also occur in a set of three or more pairs of constraints, where it is not obvious which variables are constrained to a point. Consider the constraints $\{1 \leq x-TR \leq 2, 1 \leq y-x \leq 2, 4 \leq y-TR \leq 6, 1 \leq z-TR \leq 2\}$. The first three imply $x=2, y=4$, and the volume in 3D is 0. z can take any value from $[1,2]$. However, the STP consistency algorithm can deduce which STP variables are constrained to a point.

The second approach is to add some slack to each constraint bound, so that any solution to the STP contributes some volume. This is essentially a way of adapting the “discrete” variables to make them continuous. The previous example can be modified to $\{0.9 \leq x-TR \leq 2.1, 0.9 \leq y-x \leq 2.1, 3.9$

$\leq y-TR \leq 6.1, 0.9 \leq z-TR \leq 2.1\}$. The volume is now 0.036. A single solution in 3D has volume $0.2^3=0.008$, so an estimate of the flexibility is $0.036-0.008 = 0.028$.

There remains open the problem of defining flexibility when some of the variables are not bounded, i.e. when the polyhedron represented by the constraints is not a polytope. One approach is to define an artificial domain-specific bound to all constraints. Another is to increasingly discount greater values of the variables, so that they contribute less and less to the volume computation.

Although the exact computation of the number-of-executions flexibility seems impractical, an approximate algorithm probably is tractable. Finally, more investigation into flexibility measures is warranted.

6. References

1. Tsamardinos, I., Pollack, M. E., and Ganchev, P. *Flexible Dispatch of Disjunctive Plans*. [submitted] The European Conference on Planning, 2001.
2. Barvinok A. and Pommersheim J. E. *An Algorithmic Theory of Lattice Points in Polyhedra. 1999* (<http://www.msri.org/publications/books/Book38/files/barvinok.ps.gz>).
3. Bollobás, B. *Volume Estimates and Rapid Mixing*. Flavors of Geometry, MSRI Publications, Volume 31, 1997. p. 151 - 182 (<http://www.msri.org/publications/books/Book31/files/bollobas.pdf>).
4. Dyer, M., Frieze, A. and Kannan, R. *A Random Polynomial Time Algorithm For Approximating The Volume Of Convex Bodies*, 1991.
5. Kannan, R., Lovasz, L., and Simonovitz., M. *Random walks and an $O(n^5)$ volume algorithm for convex bodies*. Random Structures Algorithms, 11:1-50, 1997.
6. Bueler, B., Enge A. and Fukuda, K. *Exact Volume Computation for Polytopes: A Practical Study*. Submitted to 'Polytopes: Combinatorics and Computation', Gunter Ziegler, editor, DMV-seminar volume, Birkhauser Verlag, 1998.

An Evaluation of the Java-based Approaches to Web Database Access

Stavros Papastavrou

*Department of Computer Science, University of Pittsburgh
Pittsburgh, Pennsylvania 15260, U.S.A.*

Panos K. Chrysanthis

*Department of Computer Science, University of Pittsburgh
Pittsburgh, Pennsylvania 15260, U.S.A.*

George Samaras

*Department of Computer Science, University of Cyprus, 75 Kallipoleos Str.
CY-1678 Nicosia, Cyprus*

Evaggelia Pitoura

*Department of Computer Science, University of Ioannina
GR-45110 Ioannina, Greece*

Received (to be inserted
Revised by Publisher)

Given the undeniable popularity of the Web, providing efficient and secure access to remote databases using a Web browser is crucial for the emerging cooperative information systems and applications. In this paper, we evaluate all currently available Java-based approaches that support persistent connections between Web clients and database servers. These approaches include Java applets, Java Sockets, Servlets, Remote Method Invocation, CORBA, and mobile agents technology. Our comparison is along the dimensions of performance and programmability. Our findings point out that best performance is not always achievable with high programmability and low resource requirements, moreover, the mobile agent technology needs to improve its programmability while giving particular emphasis on its infrastructure.

Keywords: Web Databases, Java, Mobile Agents, Servlets, CORBA, RMI, JDBC, Java Sockets

1. Introduction

Providing efficient and secure access to remote databases using a Web browser is crucial for the emerging cooperative information systems, such as Virtual Enterprises. A number of methods for Web database connectivity and integration have been proposed such as CGI scripts, active pages, databases speaking http, external viewers or plug-ins, and HyperWave¹. These methods enhance the Web server capabilities with dynamic functionality for interactive and cooperative applications to create database connections, execute queries and transactions, and generate dynamic Web

pages. However, there is an increasing interest in those that are Java-based due to the inherent advantages of Java, namely, platform independence support, highly secure program execution, and small size of compiled code, combined with a simple database connectivity interface (JDBC API) that facilitates application access to relational databases over the Web at different URLs ².

Several Java-based methods are currently available that can be used for the development of Web cooperative information systems but in the best of our knowledge, there is no quantitative comparison of them in a database context. Existing studies either primarily focused on the various server side scripting mechanisms to support database connectivity ^{3,4}, or evaluated the Java client/server communication paradigm without any database connectivity or lengthy computations ⁵. This experimental paper contributes a comparison of the six Java-based approaches, specifically, Java applets using JDBC (Applet JDBC), Java Sockets ⁶, Java Servlets ⁷, Remote Method Invocation (RMI) ⁸, CORBA ⁹, and Java Mobile Agents (JMA) ¹⁰. We focus on these methods because of their support for persistent database connections, which are essential for co-operative environments with long, and repeated data retrievals and updates. Further, these approaches are currently used as a foundation for many emerging Java-based infrastructures. For example, the servlet technology is the driving force behind Web database connectivity based on the Java Server Pages (JSP) ¹¹ and based on the Page Compile (JHTML) ¹². Another striking example to servlet technology is XSQL Servlet which processes XML documents with SQL statements inserted ¹³. XSQL servlet performs operations on XML through JDBC, thus combining the power of XML, SQL, and Java. Other examples include the CoABS Grid ¹⁴ and E*Speak ¹⁵ both of which are built on top of RMI.

For our evaluation, we used each approach to implement a Web client accessing and querying a remote database. Each approach differs in the way the client establishes connection with remote database servers with the help of a middleware and the implementation of the middleware. Depending on the way the client establishes connection with the middleware, the approaches can be classified as (1) *non-RPC* ones, that do not provide for remote method invocation mechanisms, (2) *RPC* ones with clear remote method invocation semantics, and (3) *RPC-like* ones involving mobile agent technology.

We compared the behavior of the different approaches along the two dimensions of *performance* and *programmability*. In order to get a better insight into performance, we consider both the issue of *perceived client latency*, i.e., performance observed by a single client, and *middleware scalability*, i.e., middleware ability to support multiple concurrent client requests. Perceived client latency is expressed in terms of average response time at the client site, and middleware scalability in terms of throughput at the server site under different workloads.

Development effort and programming complexity are determined by the system calls for establishing connection, submitting the query and getting back the results. Hence, we express *programmability* in terms of the number of system calls at both

the client and the server sites.

The two salient results of our study are: (1) Best performance is not always achievable with high programmability and low resource requirements, and (2) the mobile agent technology needs to improve its programmability while giving particular emphasis in its infrastructure. Our results clearly indicate that mobile agents should not be thought of and utilized as yet another simple, data shipping mechanism. Any gain in performance using mobile agents is strongly associated with their ability to significantly reduce the volume of the resulting data at the server site and therefore, cut down long data communication latencies. Hence, Java mobile agents' efficiency lies in their ability of shipping client-custom code at the server site where the high-volume data is available.

In the next section, we provide a brief introduction to Java and Java database connectivity. In Section 3, we first discuss the design principles underlying our experimental testbed and then elaborate on the implementation details of the six approaches under evaluation. In Section 4, we discuss the performance evaluation results and compare the different approaches from the programmability point of view. Section 5 concludes with a summary and future work.

2. Background: Java and Java Database Connectivity

Java¹⁶ is an interpreted, object-oriented programming language developed by Sun Microsystems Inc. with the goal of providing a distributed, secure, and portable programming language. The uniqueness of Java lies on the fact that its compiled code can run on any platform, which supports a Java runtime environment (or JVM – Java Virtual Machine). Additionally, Java programs can run in Java-enabled Web browsers in the form of applets, which are downloaded as a part of an html page. Security is achieved by restricting the execution of java applets within the context of the client's web browser, and by permitting the communication of java applets only with their originating web server. That is, Java applets are not allowed to access any system resources or communicate with any arbitrary site. Java's portability is further enhanced by several additional safety features such as the absence of pointers, automatic array bounds check and garbage collection.

Three other important characteristics of Java that are very important for building cooperative database applications, are: (1) its rich graphical interface library that supports the development of sophisticated interfaces, (2) its built-in thread library to support multithreaded operations that allow for concurrent interactions, and (3) its database connectivity interface (JDBC API) that facilitates application access to multiple relational databases over the Web at different URLs⁵.

The JDBC API is realized by various drivers, which execute under the control of a JDBC manager¹⁷. There is a driver for each particular Database Management System (DBMS). A JDBC driver can be implemented in four different ways, as shown in Figure 1. These drivers differ in two significant ways: (1) the size of their downloadable code and (2) in the way that they support multiple database

connections.

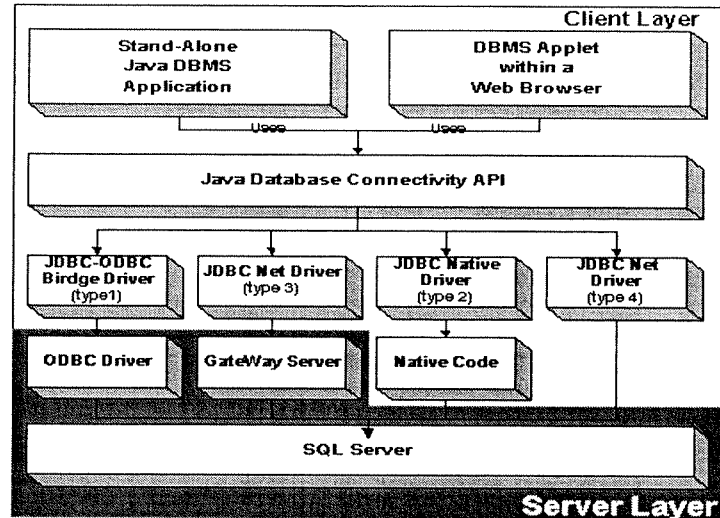


Fig. 1. Standard JDBC Methodologies

The type 1 JDBC driver, namely, the *JDBC-ODBC Bridge driver*, translates JDBC calls to ODBC ones and is suitable to access databases with only ODBC interface. A type 1 driver requires remote clients to pre-install some ODBC binary code and is not designed to be downloadable by Java applets.

The type 2 JDBC driver, the *native-API partly Java technology-enabled driver*, is written half in Java and half in a DBMS native, vendor-specific code. A client using this JDBC driver connects directly to a specific database server. In spite of the ultimate speed with which a client accesses a remote database, this method requires a-priori configuration to install DBMS native code at the client site.

The type 3 JDBC driver, the *net-protocol fully Java technology-enabled driver*, is the most flexible with Java applets. It is written entirely in Java and can be fully downloaded at run time, requiring no code pre-installation. A type 3 JDBC driver translates a client's query into an intermediate language that is converted into a vendor-specific protocol by a middle-tier *gateway*. The more vendor protocols the gateway supports, the more databases a client can access, without downloading additional drivers.

The type 4 JDBC driver is a *native-protocol fully Java technology-enabled driver* also written entirely in Java. The driver can be fully downloaded at the client at run time, and enables it to use a DBMS-specific protocol to connect directly to the remote database server. By eliminating the gateway of type JDBC 3 drivers, a type 4 JDBC driver exhibits better performance with respect to a single database access at the expense of the flexibility in accessing multiple databases without loading more than one driver. Hence, accessing multiple databases, type 4 drivers require more resources. Further, even a single type 4 driver requires the client to download

significantly more code than any other type of drivers because they include the vendor specific protocol.

3. The Experimental Testbed

We use each Java method to implement a Web client querying a remote database. Our testbed is structured along a three-tier *client/middleware/database* model. Two design principles were adopted in the selection of the various components during the development of the testbed. First, our Web clients should be lean for allowing fast downloads, and therefore increasing support for wireless clients. Second, no a-priori configuration of the Web client should be necessary to run the experiments in order to maintain portability, and therefore, support arbitrary clients. Thus, our Web client is a Java applet stored on a Web server. When the Java applet is downloaded and initialized at a client computer, queries can be issued through the applet's GUI to be executed on the remote database server (Figure 2). Our remote database server, a 3-table Microsoft Access, is on the same machine with the Web server.

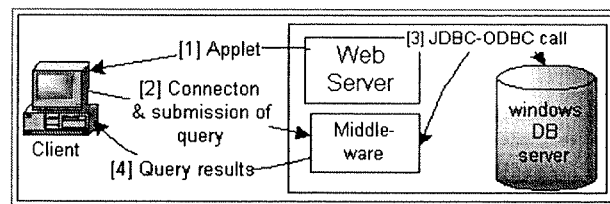


Fig. 2. Basic configuration

The role of the middleware is to accept client requests, execute them on the database server, and return the results back to the client. Due to security restrictions of Java applets, part of the middleware has to execute on the Web server machine. Recall that downloadable applets are not allowed to access any system resources or communicate with any site other than their originating web server. In our experiments, because the database server co-resides with the Web server, the entire middleware in all cases executes on the same machine. To enhance performance, if possible, the middleware connects to the database when it is activated and before any query is submitted.

Given that an Access database can only be accessed using ODBC, the middleware of all approaches except Applet JDBC, use a JDBC-ODBC bridge (type 1) driver to connect to the database. A type 1 JDBC driver cannot be used in the Applet JDBC approach in which the client applet downloads the JDBC driver, because a type 1 driver is not designed to be downloadable by Java applets. Instead we use a type 3 JDBC driver, which as opposed to type 1 and 2 drivers, can be fully downloaded at run time, requiring no code pre-installation. Further, it supports multiple vendor databases by translating clients' queries into an intermediate language that is converted into a vendor-specific protocol by a middle-tier gateway,

including JDBC calls into ODBC ones. Although, type 4 drivers in general are downloadable, in the context of our experiments in which JDBC calls need to be translated to ODBC, type 4 drivers are not appropriate because they are equivalent to type 2 ones.

In the rest of this section, we elaborate on the implementation of each approach. *Initialization phase* is the procedure for establishing database connectivity, and *execution phase* is the procedure for querying the database after the database connection is established.

3.1. Non-RPC Approaches: Java Socket and Java Servlet

Both the *Java Socket* and *Java Servlet* approaches use sockets to connect a client and the middleware program. In the Java Socket approach, sockets are created by the clients, whereas in the Servlet approach, they are created by the run-time environment.

3.1.1. The Java socket approach

In this first approach, the middleware is a stand-alone Java application server running on the Web server machine. A client collaborates with the application server by establishing an explicit socket connection.

Figure 3 illustrates the steps involved for the initial query. The applet submits the query through the socket connection to the application server, which decodes the incoming stream of data, and executes the query on the database server using the JDBC-ODBC driver. Subsequently, the application server constructs the result table by retrieving row by row from the database server. As in embedded SQL, the JDBC calls fetch one row at a time. The result table is then passed to the client applet by means of data streams.

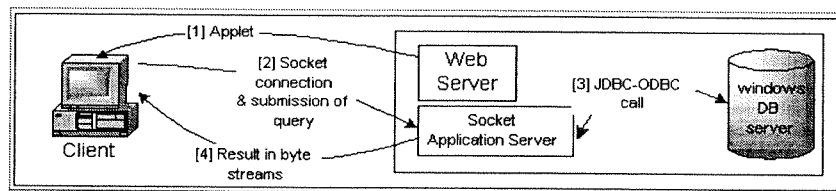


Fig. 3. The Socket approach

The cost of the first query in this approach is

- (1) Initialization phase:
 - (a) The time for the client to open a socket connection with the application server.
- (2) Execution phase:
 - (a) The time for the client to pass to the application server the data stream containing the SQL statement.

- (b) The time for the application server to execute the query, obtain the results and return them to the client.

All subsequent queries require only the execution phase.

3.1.2. Java Servlets Approach

In this approach, the middleware program is a Java servlet, which is a Java program that runs as a child process within the context of a Web server program that supports JVM and servlet engine. In our case, the *application servlet* was loaded during the Web Server start-up time. Client's queries are routed by the Web server to the application servlet, which submits them to the database server for processing. The results are assembled row by row using JDBC-ODBC calls. The results are returned to the client again through the Web server.

All queries involve both an initialization and an execution phase. Thus, the cost of any query in this approach is

- (1) Initialization phase:
 - (a) The time for the client to establish a URL connection with the Web server.
- (2) Execution phase:
 - (a) The time for the applet to invoke the application servlet passing the query as a parameter (stating explicitly the servlet name and type of operation).
 - (b) The time for the servlet to execute the query, obtain and return the result table to the client.

Figure 4 illustrates the steps required by a query in the servlet approach.

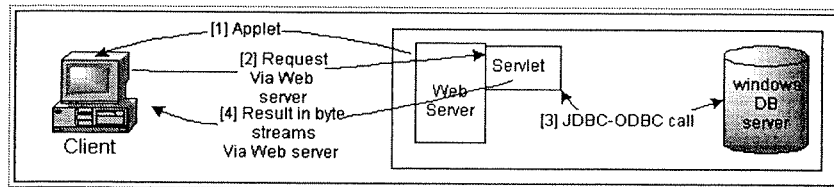


Fig. 4. Servlet approach

3.2. RPC approaches: Java RMI, CORBA, and Applet JDBC

The RPC approaches can be classified based on whether or not the client directly maintains the database connection. In the RMI and CORBA approaches, the connection is maintained by the middleware whereas in the Applet JDBC approach, by the web client.

3.2.1. The RMI approach

Java RMI is a Java application interface for implementing remote procedure calls among distributed Java objects. In RMI, the middleware consists of two objects:

the *application server* which handles the queries; and the *installer object*, which is used to start up the application server, and register it under a unique service name with the Java virtual machine running on the Web server.

Figure 5 illustrates the steps required for the initial query in the RMI approach. To establish a database connection, a client calls the RMI bind method to obtain a reference to the application server. Using this reference, the client can submit a query by calling a method on the application server passing the query as a parameter. The application server executes the query at the database server using JDBC-ODBC, assembles the results row by row, and returns the result table to the client as the return value of the method called.

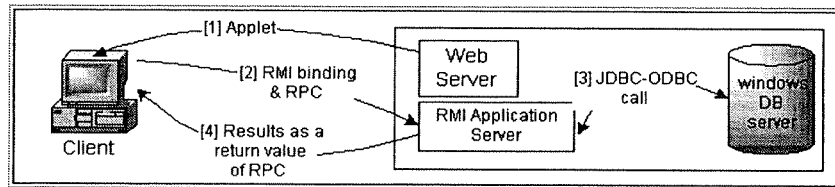


Fig. 5. RMI approach

The cost of the first query is

- (1) Initialization phase:
 - (a) The time for the applet to obtain a reference to the remote application server (bind to it) using a URL and a service name.
- (2) Execution phase:
 - (a) The time for the client to invoke a method on the application server passing the SQL statement as a parameter.
 - (b) The time for the application server to execute the SQL statement, obtain and return the results.

The time required for a subsequent query is the execution phase.

3.2.2. The CORBA approach

CORBA, the Common Object Gateway Request Broker Architecture, is an emerging distributed object standard that defines client/server relationships between objects in a common interface language. In order for a CORBA client object to utilize a CORBA server object, an implementation of CORBA's basic functionality, called the Object Request Broker (ORB), has to be loaded at both the client and the server sites. In our testbed, we use Visigenic's Visibroker for Java¹⁸, which is also included in Netscape Navigator and hence, the client does not download the ORB classes from the Web server which would have been the alternative. For security purposes, CORBA allows an applet to bind to and communicate with a remote CORBA server object only through a firewall called the IIOP (Internet Inter-ORB

Protocol) Gatekeeper, installed at the Web server machine from which the applet is downloaded.

Except from the IIOP Gatekeeper, the middleware in the CORBA approach is similar to the one in the RMI approach. There is an application server object and an installer object. The installer object in this case is also used to load the ORB, and register the application server with a unique service name with the ORB. As in the RMI, socket, and servlet approaches, the CORBA application server executes the queries using JDBC-ODBC and creates the result table retrieving row by row. The steps required for the first query are shown in Figure 6.

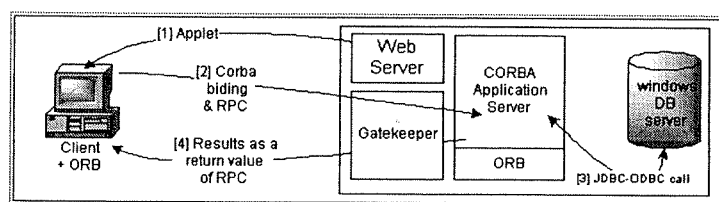


Fig. 6. CORBA approach

The cost of the first query is

- (1) Initialization phase:
 - (a) The time for the client to load and initialize core ORB classes.
 - (b) The time for the client to bind to the application server using only the service name of the application server.
- (2) Execution phase:
 - (a) The time for the client to invoke a method on the application server passing the SQL statement as a parameter.
 - (b) The time for the application server to execute the SQL, obtain and return the results to the client as the return value of the method called.

All subsequent queries require only the execution phase.

3.2.3. The Applet JDBC approach

This has been the traditional approach to web database connectivity in which Applets use directly the JDBC API. In this approach, the client applet downloads a type 3 JDBC driver and uses directly the JDBC API to connect to the database. The gateway of the type 3 driver plays the role of the middleware. After the client downloads the JDBC driver, it establishes database connectivity issuing JDBC calls on the gateway, which are then mapped to ODBC calls on the database server. Queries are submitted and their results are retrieved in the same way using JDBC calls. As in all other approaches, the resulting table is assembled by retrieving one row at a time. Figure 7 illustrates the steps involved in establishing database

connectivity and querying the remote database for the first time (initial query).

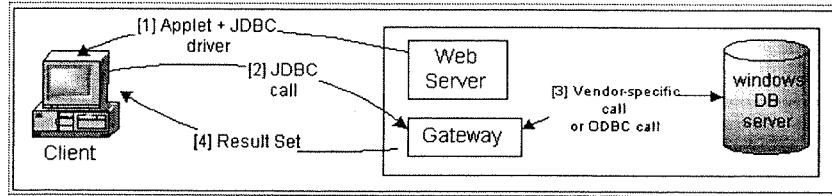


Fig. 7. The Applet JDBC approach using a type 3 JDBC driver

The cost for the first query is

(1) Initialization phase:

- (a) The time for JDBC driver to be downloaded from the Web server and initiated by the applet.
- (b) The time for the applet to establish database connection via the gateway.

(2) Execution phase:

- (a) The time for the applet to issue an SQL statement to the database and obtain the results.

All subsequent queries require only the execution phase.

3.3. RPC-like approach: Java Mobile Agents (JMA)

Finally, in this subsection, we describe the approach of using mobile agents to achieve Web database connectivity, and specifically, the best of the three proposed variants^{19,20}. Mobile agents are processes capable of pausing their execution on one machine, dispatching themselves on another machine and resuming their execution on the new machine. The idea in the JMA approach is to use one or more mobile agents to implement the middleware and carry out the requests of the client.

For our experiments, we used *Aglets*²¹, for two reasons: (a) availability of code, and (b) support for hosting mobile agents within applets without significant overhead based on our prior experience with their use. Aglets can be fired from within a special applet, called the *FijiApplet* that provides an aglet execution environment similar to the general stand-alone aglet runtime environment called the *Tahiti Server*.

In the JMA approach, the middleware consists of three components: The *DBMS-aglet*, the (*Stationary*) *Assistant-aglet* and the *Aglet Router*. The DBMS-aglet can connect to a database and submit queries. Each database server is associated with an Assistant-aglet identified by a unique aglet ID and a URL. An Assistant-aglet provides the information necessary for a DBMS-agent to load the appropriate JDBC driver and connect to the database server. An Aglet Router is required to route aglets and messages, dispatched from a FijiApplet to any destination, and vice

versa, because of the Java security restrictions.

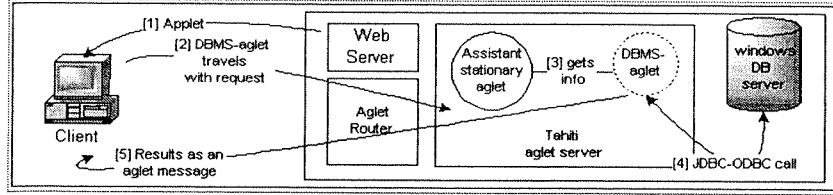


Fig. 8. Mobile agents approach configuration (message variation)

When the user enters the first query (Figure 8), the client applet (an extension of the FijiApplet) creates a DBMS-aglet with a specific URL-based itinerary (travel plan) and the specified query. The DBMS-aglet travels through the aglet router to the database server. Upon its arrival, the DBMS-aglet communicates with the Assistant-aglet to retrieve information on the database and drivers, loads the JDBC-ODBC driver, connects to the database server, executes the client's request and assembles the resulting table. After returning the query result in a message to the client, the DBMS-aglet remains connected to the database server, waiting for a message with new requests from the client. This message passing is implemented implicitly as an RPC invocation from the client applet on the dispatched mobile agent.

The cost of the initial query is

(1) Initialization phase:

- (a) The time for the client to create the DBMS-aglet
- (b) The time for the client to initialize the DBMS-aglet (SQL statement, itinerary, etc.)
- (c) The time for the DBMS-aglet to travel to the remote database server
- (d) The time for the DBMS-aglet to negotiate with the assistant aglet
- (e) The time for the DBMS-aglet to establish connection with the database

(2) Execution phase:

- (a) The time for the DBMS-aglet to query the database and send the results to the client using a message.

All subsequent requests require only one message from the client to DBMS-aglet, which includes the new SQL statement, plus the execution phase.

4. Evaluation of Approaches

In this section, we will first discuss our experiments measuring the *perceived client latency*, expressed in terms of average response time at the client site. Then, we will discuss *middleware scalability* and express it in terms of average throughput at the web server site under different workloads. Finally, we will compare the different approaches in terms of their programmability.

Our experiments consider both small interactions and heavy cooperations. Small interactions typically involve a small size query result (128 bytes) while heavy cooperation involves a wide range of query result sizes. Given our interest to support both mobile clients and clients over a wide-area network with relatively slow communication links (limited bandwidth), we conducted our experiments on a wireless 1.2Mbps LAN of Pentium PCs. We used Netscape Navigator as the Web client's Java-enabled browser. For each approach, a sufficient number of runs were performed to obtain statistically significant results.

4.1. Client Perceived Latency

4.1.1. Small Interactions

We measured the response time (a) of the first query and (b) of subsequent queries (Figure 9). Short-duration interactions consist of a single query as opposed to long duration ones. The execution of the first query differs from the subsequent ones because it incurs the overhead of establishing the connection between the client and the remote database.

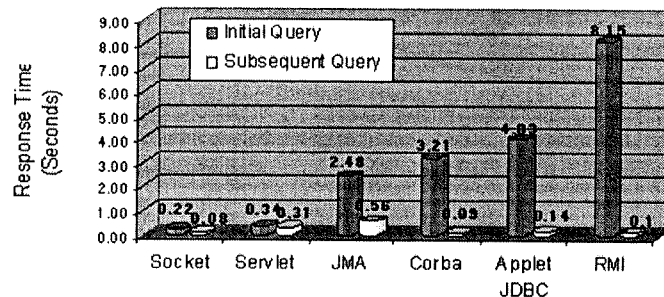


Fig. 9. Performance of all approaches for 128 bytes result size

For the first query (short-duration interactions), the non-RPC approaches have by far the lowest response time. This can be explained by the fact that their initialization phase does not engage any special package loading by the client. Compared to the Socket approach, the Servlet approach performs slightly worse because (a) the communication between the client and the servlet is marshaled by the Web server, and (b) by executing as a Web server thread, the servlet receives less CPU time than the socket application server. Thus, servlets respond slower to requests and take more time to assemble the query results.

From the other approaches, the JMA approach offers the best performance for a single query. Significant part of its cost (around 2 sec) is due to the process of dispatching the DBMS-aglet from the client applet to the aglet router on the Web server and from there to the database server. In the case of the CORBA approach, the first query is slightly more expensive than the one in the JMA approach because

of the overhead of initializing the necessary ORB classes and the binding to the application server. This overhead is quite significant (around 2.20 sec). Following the CORBA approach is the Java JDBC approach in which the response time of the first query is increased by a considerable amount of time by the downloading of the JDBC driver.

To our surprise, the RMI approach performs by far the worst for the first query. We expected the RMI approach to exhibit better performance because, as opposed to the other RPC approaches, it does not involve the loading of any specific package. The only way to explain this is to attribute the increased response time to the interpreted method of RMI calls when binding the client applet to the application server. CORBA compilers create hard-coded encoding/decoding routines for marshaling of objects used as RPC parameters, whereas RMI uses object serialization in an introspective manner. This means that (a) RMI encodes additional class information for each object passed as a RPC parameter, and (b) marshaling is done in an interpreted fashion. Consequently, RMI remote calls are more demanding in terms of CPU time and size of code transmitted, a fact that we observed in all our experiments.

For subsequent queries (long-duration interactions), the performance of the CORBA and RMI approaches dramatically improves, and becomes close to the best performance exhibited by the Socket approach. The reason is that the client applet is already bound to the application server and only a remote procedure call is required to query the database. For a similar reason, the JDBC applet approach also exhibits a significant performance improvement for subsequent queries.

Having the DBMS-aglet already connected to the remote database and ready to process a new query, the JMA approach also improves its response time for subsequent queries. However, this response time is the worst of all other approaches. We attribute this to two reasons: (1) the two required messages to implement subsequent queries have to be routed through the aglet router, and (2) a mobile agent is not a stand-alone process and it does not receive full CPU time.

Finally, the Servlet approach improves slightly its performance although the steps for executing any query are the same. This improvement is due to the fact that any subsequent connection between the client and the Web server require less time because its URL has already been resolved in the initial query.

In order to better illustrate the *overall performance* of each approach, we plotted in Figure 10 the average time required by each approach for a number of consecutive queries. It is clear that the socket approach is the most efficient for both short- and long-duration interactions. This is not a surprise since all other approaches are built on top of sockets. Both the Servlet and the JMA approaches scale very badly. The CORBA, JDBC applet, and RMI approaches appear to scale well, however, the RMI approach appears less attractive due to its worst performance for initial

queries.

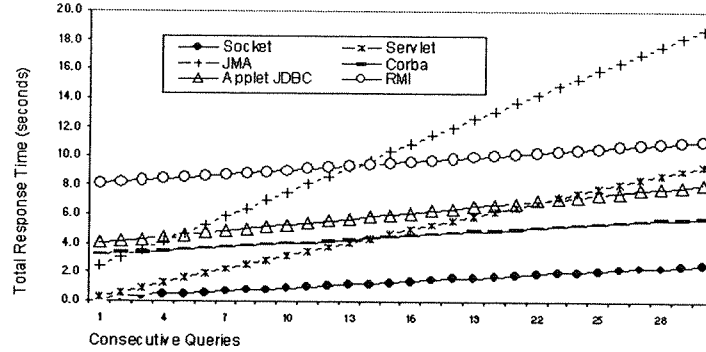


Fig. 10. Average performance for up to 30 consecutive queries (128 bytes of result size)

4.1.2. Heavy Cooperation

In order to evaluate heavy cooperation, we adjusted the size of the query result from 128 bytes (2 tuples) to 64 kilobytes (1000 tuples) by changing the complexity of the SQL statement issued through the client applet. Query result size directly affects the response time in two ways: (1) in the amount of time spent for the query to execute, and (2) in the transport time for the results to reach the client. The latter includes the overhead in assembling the result table. For these experiments, we also measured response times of first and subsequent queries. Tables 1 and 2 summarize these results. In both cases, each approach exhibited similar sensitivity, as shown in Figures 11 and 12.

Table 1. Heavy Cooperation. Average response time in seconds - initial query

Approach/Result Size	128B	512B	1KB	5KB	10KB	20KB	64KB
Socket	0.22	0.25	0.44	0.70	1.44	4.08	29.08
Servlet	0.34	0.35	0.48	1.82	7.34	21.47	189.80
CORBA	3.21	3.31	3.32	3.36	4.22	6.96	34.93
RMI	8.15	8.37	8.50	10.67	11.66	15.73	47.03
JMA	2.48	2.60	2.61	3.11	7.86	20.57	116.12
Applet JDBC	4.03	4.04	4.12	8.80	17.32	40.63	248.00

Table 2. Heavy Cooperation. Average response time in seconds - subsequent query

Approach	Result Size	128B	512B	1KB	5KB	10KB	20KB	64KB
Socket		0.08	0.10	0.26	0.51	1.34	3.97	28.73
Servlet		0.31	0.32	0.39	1.73	7.15	21.26	140.00
CORBA		0.09	0.10	0.28	0.53	1.41	4.01	30.00
RMI		0.10	0.11	0.30	1.12	2.54	6.45	38.07
JMA		0.56	0.64	0.73	2.00	6.40	18.57	91.00
Applet JDBC		0.14	0.30	0.68	5.23	14.86	37.69	227.00

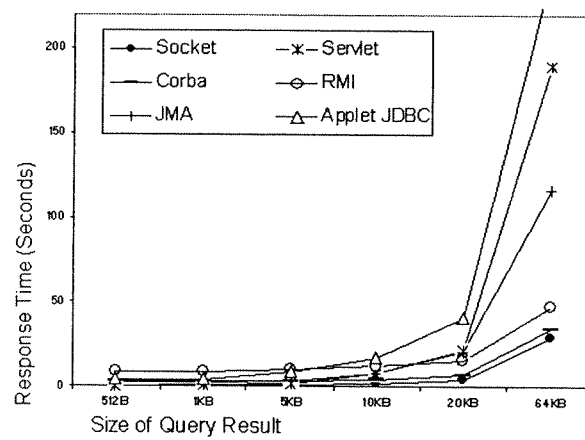


Fig. 11. Initial Query

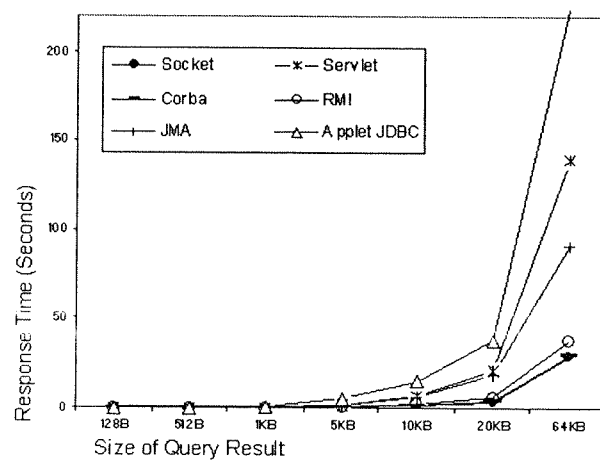


Fig. 12. Subsequent Query

The first observation is that the average response times of Java JDBC applet, Servlet and JMA approaches increase exponentially with query result sizes larger than 20KB. The applet JDBC approach performs by far the worst for increased result size. This can be explained by the fact that in JDBC, rows from a query result are retrieved one at a time. Specifically, to retrieve one row from the query result, the client must call a method on a Java ResultSet object, which is mapped on the remote database server through the Gateway. Consequently, for a large size of query result, a large number of those remote calls have to take place. For example, to retrieve the 64KB result table, the client invokes 1000 remote calls on the database server via the gateway. These large number of sequential message exchanges not only increase dramatically the response time but also increase the Internet traffic.

The bad scaling of the JMA approach can be explained in the same way as the bad performance of the Servlet approach. Both mobile agents and servlets do not execute as stand-alone processes, and therefore, they do not receive full CPU time and heavily depend on the supporting execution environment. The other RPC approaches exhibit acceptable performances (close to linear for sizes above 20KB) with the CORBA approach being slightly better. As indicated above, the implementation of RPC calls in CORBA is much faster compared to the RMI's one.

Figures 11 and 12 clearly suggest that Java mobile agents are not practical to be used as a simple communication mechanism compared to other approaches like CORBA and RMI. As mentioned in the introduction, it is not appropriate to consider Java mobile agents as yet another data shipping mechanism, but rather as a function shipping one. The mobile agents effectiveness lies in their ability of performing data-reduction operations such as aggregates and data mining that cut down the size of the result to be returned to the client. Thus, JMA are feasible only in cases where client-custom code can be shipped and execute at the server site to perform additional filtering on the query result, as in the case of the MOCHA database middleware system²². Our results (see, Tables 1 and 2) indicate that for query results of 20KB or less, the JMA approach becomes effective when they can achieve result reduction by some factor greater than 5. For example, for 20KB subsequent queries, in CORBA the client perceived latency is 4.01 sec, which is two times the latency for a 5-times reduced (i.e., 4KB) result in JMA which is less than 2 sec. For query result sizes greater than 20KB, a reduction of 2 is needed to make the JMA approach feasible.

4.2. Middleware Scalability

Although practically important, the perceived client latency experiments do not fully measure each approach's effectiveness because they do not capture the effects of concurrent clients on the corresponding middleware performance. For this reason, we measure the average throughput for each approach in terms of average number of client requests that can be served per second by the corresponding middleware.

For the CORBA and RMI approaches, concurrent RPC calls from Web clients to the application server object are handled by the underlying ORB and by the Java virtual machine at the server site respectively. For the servlet approach, concurrent Web client connections to servlets are handled by the Web server itself and are persistent. For the applet JDBC approach, JDBC calls from concurrent Web clients are handled by the gateway at the server site over persistent connections. In the socket approach, each client is served by a separate thread selected from a pool of threads maintained by the socket application server. Similar to the other approaches, the socket connection between the client and its server thread is persistent to increase performance. Finally, for the JMA approach, one mobile agent is dispatched from each concurrent Web client to the server site. In this approach, however, the connection between a clients and its mobile agent is not persistent.

For our experiments, we assumed a wide range of concurrent Web clients (multi-programming level or MPL) repeatedly issuing requests with a random 'think' time between successive ones. Since we were only interested in the performance of the middlewares in a steady state, the Web client programs used in these measurements had already issued the initial query. Figures 13, 14, and 15 illustrate the average throughput for each approach in requests served per second for MPLs between 1 and 25, and for a query result size of 1, 5, and 20 kilobyte, respectively.

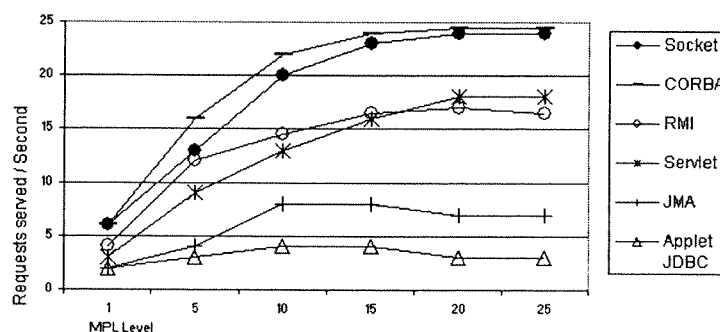


Fig. 13. 1K Query result size

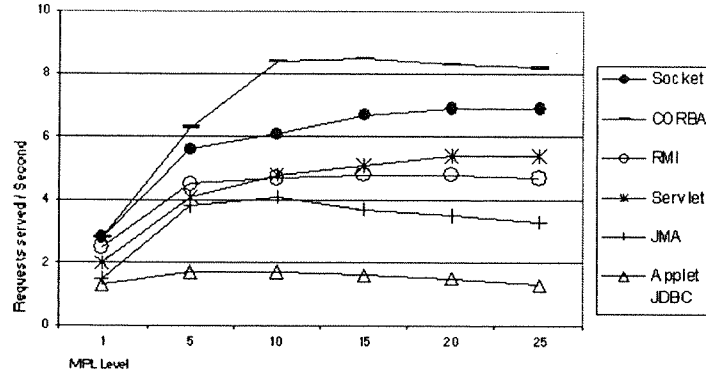


Fig. 14. 5K Query result size

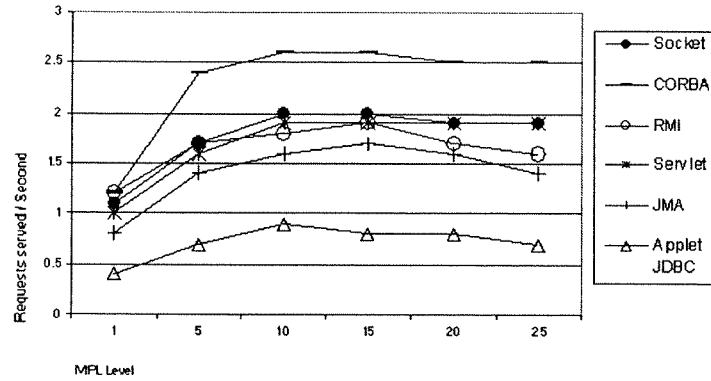


Fig. 15. 20K Query result size

These results are generally in agreement with our results of the client perceived latency. Perhaps the only exception is with the performance of the socket approach, which was expected to outperform all the other ones. However, the CORBA approach exhibited higher throughput than the socket approach for increasing size of query result. Our explanation is that the ORB bus optimizes multiple connections originating from the same sites whereas in our basic socket application server this is not the case. Multiple socket connections are competing for network bandwidth resulting in a major performance penalty in case of low bandwidth wireless communication links.

Another interesting observation is the crossover occurring between the servlet and RMI approaches for high MPL in all the query result sizes. For low MPL, the RMI approach exhibits higher throughput than the servlet approach. For high MPL, that is, a large number of concurrent clients, it seems that the interpreted way RMI handles RPC arguments contributes significantly to the system overloading,

causing the throughput to drop even lower than the servlet approach.

For any query result size, the applet JDBC approach performed the worse. These results exemplify how costly is the way this approach assembles the query results. Recall that the retrieval of each row of the result table requires a separate remote JDBC procedure call by the client. Clearly, the applet JDBC approach is not appropriate for any wireless and mobile environment.

The poor performance exhibited by the JMA approach raises the important issue of site overloading and the effectiveness of an agent-hosting environment (the Tahiti server in our case). With mobility, more agents can arrive at a site with varying resource requirements than the server site can effectively handle. Hence, this competition for resources, not only affects the performance of the visiting agents but also impacts any other service provided at the hosting site. In our case, both the Web and database server were affected, degrading their performance.

Our last observation is that with the increase of query result size, the throughput gap among the all approaches shrinks. For query result sizes of 20K and higher, the throughput of the socket, RMI, and servlet approaches is practically the same. This can easily be explained because in all approaches except the applet JDBC one, the procedure of assembling the query results is exactly the same and is carried out entirely at the server site. It invokes a JDBC-ODBC call for each row in the result table. Having an increasing size of query result, and therefore more result rows, more CPU and memory resources are needed at the server site, leading also faster to thrashing.

4.3. Programmability Comparison

In this section, we compare the different approaches in terms of development effort. Our goal is to understand if there is any correlation or trade-off between performance and programming complexity. To quantify the development effort, we use the number of required system calls. The number of systems calls used in each approach is, in some sense, analogous to the number of code lines implementing each approach. Table 3 shows the total number of system calls required for each approach.

A first observation is that the development effort of the client is related to the level of abstraction of communication between the client and the middleware, in general, and the naming scheme used to identify the database services to establish communication, in particular. Not surprisingly, the RPC approaches involve less complex APIs, more transparent client/server communication and hence exhibit high programmability. All non-RPC approaches, including the JMA approach (the RPC-like one), require more development effort and hence have low programmability.

A second observation is that despite the fact that the JMA approach supports RPC-like communication, it exhibits the *lowest* programmability as indicated by the largest number of system calls required. Most of these system calls are used to

Table 3. Programmability of the approaches

Notes	Socket	Servlet	CORBA	RMI	Applet JDBC	JMA
Total Number of System Calls	29	25	15	12	6	29
Establish Connection						
At the Client	7	11	2	1	3	11
At the Middleware	11	3	8	6	0	11
Submit Query and Get Results						
At the Client	3	3	1	1	3	2
At the Middleware	8	8	4	4	0	5
Client Execution Code	6K	6K	23K	9K	50K	27K
Programmability	Low	Low	High	High	High	Low

construct, maintain and execute the URL-based itinerary. It seems this is a result of the *weak mobility* model currently supported by most mobile agents implementations^{23,24}. In the weak mobility model, the agent program is responsible to handle the details of all mobility functions. In a *strong mobility* model²⁵, agents could migrate at any point during their execution and resume their execution on a different site without any special preparation for migration.

Finally, the level of programmability does not correspond to the size of the client executable code. Interestingly, the Non-RPC approaches, namely, Java Socket and Servlet, support the smallest client size (6K). On the other hand, the Applet JDBC has the largest client size of 50K: the Java applet is 6K and the JDBC driver is 46K. The JMA approach is the second most resource demanding approach after Applet JDBC with 27K: Java applet 10K, FijiApplet 10K and DBMS-Aglet 7K.

5. Conclusions and Future Work

In this experimental paper, we have implemented, evaluated, and compared all currently available Java-based approaches that support persistent Web database connectivity. Our comparison proceeded along the lines of the performance of query processing and of the programmability of each approach.

The results of our comparison showed that:

- The CORBA approach offers high programmability and hence, is easy to develop, while its overall performance with respect to both client perceived latency and middleware scalability is comparable to the best performing approach that employs sockets. On the other hand, it is resource demanding and leads to clients with big footprints. Therefore, the CORBA approach offers the best promise for the development of large and complex Web applications, in particular, cooperative interactions involving multiple queries of varying result sizes and without resource constraints.
- For small interactions, typically involving a single query, few clients and envi-

ronments with resource-starved clients, the socket and servlet approaches should be considered. These approaches yield a Web client with the smallest footprint, just 6 Kbytes. Both approaches have low programmability but relative to each other, the servlet approach is easier to develop than the socket approach. On the other hand, the socket approach exhibits better performance than servlets. Thus, for such interactions, the trade-off between these approaches is performance and programmability.

- Clearly, the best performance is not always achievable with high programmability and low resource requirements. The surprising example here is the RMI approach due to its popularity which can be attributed to its high programmability. Despite RMI's popularity in the fixed network environments, its implementation needs to be optimized before it can effectively be utilized in mobile and wireless environments.
- Finally, the JMA approach cannot support interactions that require movement or exchange of large amounts of data such as a large number of consecutive queries with increased size of query result. Hence, it is necessary to develop more efficient mobile agent infrastructures, if the full potential of mobile agents is to be explored.

The recent advancements of the Web technology and mobile computing led to a renewed interest on mobile agents technology. Given this renewed interest, our study provided an insight to potential scalability problems with the currently available mobile agent implementations. Given that CORBA offers the best overall performance, as part of our future work, we will investigate the possibility of merging mobile agents and the CORBA technology in order to facilitate a scalable and efficient JMA-based Web database connectivity. Our goal is not to make mobile agents yet another communication paradigm but instead a more effective distributed computing one.

Acknowledgments

The authors thank the anonymous reviewers for their very helpful suggestions. This work was partially supported by NSF IIS-9812532, and AFOSR F49620-98-1-043 awards.

References

1. H. Maurer, *Hyperwave: The Next Generation Web Solution* (Addison-Wesley, 1996).
2. B. Jepson, *Java Database Programming* (Wiley Computer Publishing, 1997).
3. G. Ehmayr, G. Kappel, and S. Reich, Connecting Databases on the Web: A Taxonomy of Gateways, in *Proc. 8th DEXA International Conference and Workshops*, September 1997, 1–15.
4. A. Lambrinidis, and N. Rousopoulos, Generating dynamic content at database-backed web server: cgi-bin vs mod_perl, *Sigmod Record*, **29(1)** (2000), 26–31.

5. R. Orfali, and D. Harkley, *Client Server Programming with Java and CORBA* (Wiley Publishing, 1998).
6. Sun Microsystems Inc., *Java Sockets Documentation*, (<http://java.sun.com/docs>).
7. J. Goodwill, *Developing Java Servlets* (Sams Publishing, 1999).
8. T. B. Downing, *Java RMI: Remote Method Invocation* (IDG Books Worldwide, 1998).
9. Object Management Group. The Common Object Request Broker: Architecture and specification (1998).
10. D. Chess, B. Grosz, C. Harrison, D. Levine, C. Parris, and G. Tsudik, Itinerant Agents for Mobile Computing, *IEEE Personal Communications*, **2(5)** (1995) 34–49.
11. Sun Microsystems Inc., *Java Server Pages: Dynamically Generated Web Content* (<http://java.sun.com/products/jsp/>).
12. Sun Microsystems Inc., *Java Web Server: Create and Deploy Dynamic Web Pages in a Snap* (<http://www.sun.com/software/jwebserver/>).
13. Oracle, *XSQL* (<http://technet.oracle.com/free>).
14. DARPA, ninite The CoABS Grid (<http://coabs.globalinfotek.com/>).
15. Hewlett-Packard, *E*Speak* (<http://e-speak.hp.com>).
16. E. Anuff, *Java Sourcebook* (Wiley Publishing, 1996).
17. Sun Microsystems Inc., *JDBC drivers*, (<http://java.sun.com/products/jdbc/drivers.html>).
18. Borland Inc., *Visibroker for Java V.2.0* (<http://www.inprise.com/visibroker>).
19. S. Papastavrou, G. Samaras, and E. Pitoura, Mobile Agents for WWW Distributed Database Access, in *Proc. 14th IEEE Int'l Conf. on Data Engineering*, March 1999, 228–237.
20. S. Papastavrou, G. Samaras and E. Pitoura, Mobile Agents for World Wide Web Distributed Database Access, *nineit IEEE Transactions on Knowledge and Data Engineering*, **12(5)** (2000) 802–820.
21. IBM Japan Research Group, *Aglets Workbench* (<http://www.trl.ibm.co.jp/aglets>).
22. M. Rodriguez-Martinez, and N. Rousopoulos, MOCHA: A Self-Extensible Database Middleware System for Distributed Data Sources, in *Proc. ACM Sigmod Conference*, May 2000, 213–224.
23. D. Kotz, Robert S. Gray, S. Nog, D. Rus, S. Chawla, and G. Cybenko, AGENT TCL: Targeting the Needs of Mobile Computers, *IEEE Internet Computing*, **1(4)** (1997) 58–67.
24. D. Wong, N. Paciorek, T. Walsh, J. DiCeie, M. Young, and B. Peet. Concordia: An infrastructure for Collaborating Mobile Agents, in *Proc. 1st Int'l Workshop on Mobile Agents*, 1997, 367–376.
25. N. Suri, J.M. Bradshaw, M. R. Breedy, P.T. Groth, G.A. Hill, and R. Jeffers. Strong Mobility and Fine-grained Resource Control in NOMADS, in *Proc. 2nd Int'l Symposium on Agent Systems and Applications and 4th Int' Symposium on Mobile Agents*, September 2000, 1882:2–15.

An Evaluation of the Java-based Approaches to Web Database Access*

Stavros Papastavrou¹, Panos Chrysanthis¹, George Samaras², Evaggelia Pitoura³

¹ Dept. of Computer Science, University of Pittsburgh
{stavrosp, panos}@cs.pitt.edu

² Dept. of Computer Science, University of Cyprus
cssamara@cs.ucy.ac.cy

³ Dept. of Computer Science, University of Ioannina
pitoura@cs.uoi.gr

Abstract. Given the undeniable popularity of the Web, providing efficient and secure access to remote databases using a Web browser is crucial for the emerging cooperative information systems and applications. In this paper, we evaluate all currently available Java-based approaches that support persistent connections between Web clients and database servers. These approaches include Java applets, Java Sockets, Servlets, Remote Method Invocation, CORBA, and mobile agents technology. Our comparison is along the important parameters of *performance* and *programmability*.

1 Introduction

Providing efficient and secure access to remote databases using a Web browser [2,6] is crucial for the emerging cooperative information systems, such as Virtual Enterprises. A number of methods for Web database connectivity and integration have been proposed such as CGI scripts, active pages, databases speaking http, external viewers or plug-ins, and HyperWave [9]. These methods enhance the Web server capabilities with dynamic functionality for interactive and cooperative applications to create database connections, execute queries and transactions, and generate dynamic Web pages. However, there is an increasing interest in those that are Java-based due to the inherent advantages of Java, namely, platform independence support, highly secure program execution, and small size of compiled code.

Several Java-based methods are currently available that can be used for the development of Web cooperative information systems but in the best of our knowledge, there is no quantitative comparison of them in a database context. Existing studies either primarily focused on the various server side scripting mechanisms to support database connectivity (e.g., [8, 12]), or evaluated the Java client/server

* This work was partially supported by NSF IRI-9502091 and IIS-9812532, and AFOSR F49620-98-1-043 awards.

communication paradigm without any database connectivity or lengthy computations (e.g., [14]). This experimental paper contributes a comparison of the six Java-based approaches that support persistent database connections, specifically, Java applets using JDBC (Applet JDBC), Java Sockets, Java Servlets, Remote Method Invocation (RMI), CORBA, and Java Mobile Agents (JMA). We focus on these methods because of their support for *persistent* database connections, which are essential for cooperative environments with long, and repeated data retrievals and updates.

For our evaluation, we used each approach to implement a Web client accessing and querying a remote database. Each approach differs in the way the client establishes connection with remote database servers with the help of a middleware and the implementation of the middleware. Depending on the way the client establishes connection with the middleware, the approaches can be classified as (1) *non-RPC* ones, that do not provide for remote method invocation mechanism, (2) *RPC* ones with clear remote method invocation semantics, and (3) *RPC-like* ones involving mobile agent technology.

We compared the behavior of the different approaches along the following two important parameters: (1) *performance* expressed in terms of response time under different loads, and (2) *programmability* expressed in terms of the number of system calls at the client and the server site. The two salient results of our study are: (1) Best performance is not always achievable with high programmability and low resource requirements, and (2) the mobile agent technology needs to improve its programmability while giving particular emphasis in its infrastructure.

In the next section, we provide a brief review of Java and Java database connectivity. In Section 3, we first discuss our experimental testbed and then elaborate on the implementation details of the six approaches under evaluation. In Section 4, we discuss our performance evaluation results whereas in Section 5, we compare the different approaches from programmability point of view.

2 Background: Java and Java database connectivity

Java [17,1] is an object-oriented programming language designed to support the development of distributed, secure, and portable applications. The uniqueness of Java lies on the fact that its compiled code can run on any platform, which supports a Java runtime environment. Further, Java programs can run in Java-enabled Web browsers in the form of applets, which are downloaded as part of an html page. Security is achieved by restricting the execution of applets within the context of the client's web browser, and by permitting the communication of applets only with their originating web server. That is, Java applets are not allowed to access any system resources or communicate with any arbitrary site. Java's portability is further enhanced by other safety features, such as the absence of pointers, and automatic array bound check.

Two features of Java, important for building cooperative database applications, are: (a) its graphical interface library that supports the development of sophisticated

interfaces, and (2) its database connectivity interface (JDBC API) that facilitates application access to relational databases over the Web at different URLs [11].

The JDBC API is implemented by various drivers, executing under the control of a JDBC manager [19]. A JDBC driver can be implemented in four different ways, as shown in Figure 1. These drives differ in two significant ways: (1) the size of their downloadable code, and (2) in the way they support multiple database connections.

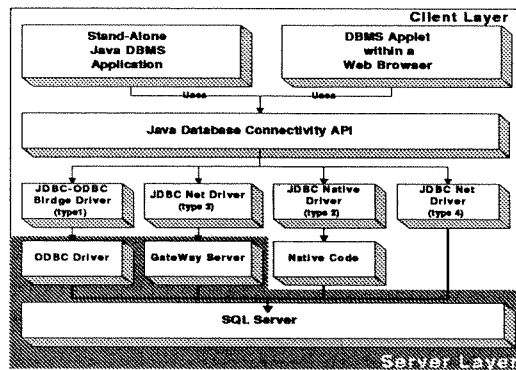


Figure 1: Standard JDBC Methodologies

The type 1 JDBC driver, namely, the *JDBC-ODBC Bridge* driver, translates JDBC calls to ODBC ones and is suitable to access databases with only ODBC interface. A type 1 driver requires remote clients to pre-install some ODBC binary code and is not designed to be downloadable by Java applets.

The type 3 JDBC driver, the *net-protocol fully Java technology-enabled* driver, is the most flexible with Java applets. It is written entirely in Java and can be fully downloaded at run time, requiring no code pre-installation. A type 3 driver translates a client's query into an intermediate language that is converted into a vendor-specific protocol by a middle-tier gateway. The more vendor protocols the gateway supports, the more databases a client can access, without downloading additional drivers.

3 The Experimental Testbed

We use each Java method to implement a Web client querying a remote database. Our testbed is structured along a three-tier *client/middleware/database* model. Two design principles were adopted in the selection of the various components during the development of the testbed. First, our Web clients should be lean for allowing fast downloads, and therefore increasing support for wireless clients. Second, no a-priori configuration of the Web client should be necessary to run the experiments in order to maintain portability, and therefore, support arbitrary clients.

Our Web client program is a Java applet, installed on a Web server along with an html page. Every experiment was initiated by pointing to the html page from a remote computer. After the Java applet was downloaded and initialized at the client computer, database connectivity was established, and queries were issued through the applet's GUI to be executed on the remote database server. Our remote database system, a 3-table Microsoft Access, was on the same machine with the Web server.

The role of the middleware is to accept client requests, execute them on the database server on behalf of the client, and return the results back to it. Due to security/communication restrictions of Java applets, part of the middleware in all approaches has to execute on the Web server machine. In the experiments reported here, because the database server co-resides with the Web server, the entire middleware in all approaches executes on the same machine. Given that an Access database can only be accessed using ODBC, the middleware of all approaches except Applet JDBC, use a JDBC-ODBC (type 1) driver to connect to the database. In the Applet JDBC approach, a type 3 JDBC driver is used whose gateway converts the JDBC calls into ODBC ones. To improve performance, the middleware attempts to connect to the database server when it is activated and before any client request is submitted.

In the rest of this section, we elaborate on the implementation of each approach. *Initialization phase* is the procedure for establishing database connectivity, and *execution phase* is the procedure for querying the database after database connection is established.

3.1 Non-RPC Approaches: Java Socket and Java Servlet

Both the *Java Socket* and *Java Servlet* approaches use sockets to connect a client and the middleware program. In the Java Socket approach, sockets are created by the clients, whereas in the Servlet approach, are created by the run-time environment.

3.1.1 The Java socket approach. In this first approach, the middleware is a stand-alone Java application server running on the Web server machine. The client collaborates with the application server by establishing an explicit socket connection [18]. Figure 2 illustrates the steps involved for the first query. The applet submits the query through the socket connection to the application server, which decodes the incoming stream of data, and executes the query on the database server. The result table is then passed to the client applet again by the means of data streams.

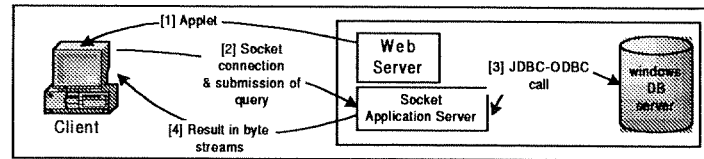


Figure 2: The Socket approach

The cost of the first query in this approach is

1. Initialization phase:
 - A. The time for the client to open a socket connection with the application server.
2. Execution phase:
 - A. The time for the client to pass to the application server the data stream containing the SQL statement.
 - B. The time for the application server to execute the query, obtain the results and return them to the client.

All subsequent queries require only the execution phase.

3.1.2 Java Servlets Approach. In the Java Servlet approach, the middleware program is a Java servlet [5], which is a Java program that runs as a child process within the context of a Web server program. The Web server is responsible for loading, maintaining, and terminating servlets. In our case, servlets were loaded during the Web Server start-up time.

Client's queries are routed by the Web server to a servlet, which submits them to the database server for processing. The results are returned to the client again through the Web server. All queries involve both an initialization and an execution phase. Thus, the cost of any query in this approach is

1. Initialization phase:
 - A. The time for the client to open a URL connection with the Web server.
2. Execution phase:
 - A. The time for the applet to invoke, through the Web server, the corresponding servlet passing the SQL statement as a parameter (stating explicitly the servlet name and type of operation).
 - B. The time for the servlet to execute the request, obtain and return the entire result table to the client.

3.2 RPC approaches: Java RMI, CORBA, and Applet JDBC

The RPC approaches can be classified based on whether or not the client directly maintains the database connection. In the RMI and CORBA approaches, the connection is maintained by the middleware whereas in the Applet JDBC approach, by the web client.

3.2.1 The RMI approach. Java Remote Method Invocation (RMI) [4] is a Java application interface for implementing remote procedure calls between distributed Java objects. In RMI, the middleware consists of two objects: The first object is the *application server* which is responsible for handling requests by allowing clients to remotely invoke methods on it. The second object is the *installer object*, which is used to start up the application server, and register it under a unique service name with the Java virtual machine running on the Web server.

Figure 3 shows the steps for the first query. The client calls a bind method of the RMI to obtain a reference of the application server. The parameters of this bind method are the URL of the machine on which the application server object was registered, and the unique service name with which it was registered. Using this reference, the client calls a method on the remote application server passing the query as a parameter. The application server executes the query at the database server, and returns the result table to the client as the return value of the method called.

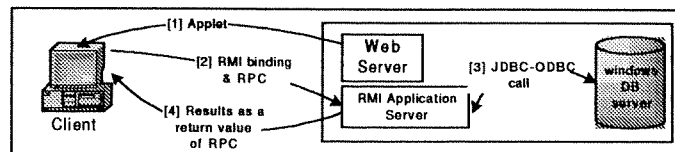


Figure 3. RMI approach

The cost of the initial query is

1. Initialization phase:
 - A. The time for the applet to obtain a reference to the remote application server (bind to it).
2. Execution phase:
 - A. The time for the client to invoke a method on the application server passing the SQL statement as a parameter.
 - B. The time for the application server to execute the SQL statement, obtain and return the results.

The time required for a subsequent query is the execution phase.

3.2.2 The CORBA approach. CORBA, the Common Object Gateway Request Broker Architecture [13], is an emerging distributed object standard that defines client/server relationships between objects in a common interface language. Unlike

RMI, CORBA objects can be implemented in any programming language. In order for a CORBA client object to utilize a CORBA server object, an implementation of CORBA's basic functionality, called the Object Request Broker (ORB), has to be loaded at both the client and the server sites. In our testbed we use Visigenic's Visibroker for Java [20], which is also included in Netscape Navigator and hence, the client does not download the ORB classes from the Web server which would have been the alternative. For security purposes, CORBA allows an applet to bind to a remote CORBA server object only through a firewall called the IIOP (Internet Inter-ORB Protocol) Gatekeeper [21], installed at the Web server machine from which the applet is downloaded. That is, the IIOP Gatekeeper is responsible for routing the client's calls on the loaded ORB, and to the application server.

Except from the IIOP Gatekeeper, the middleware in the CORBA approach is similar to the one in the RMI approach. There is an application server object and an installer object. The installer object in this case is also used to load the ORB, and register the application server with a unique service name with the ORB.

The steps required for the first query are shown in Figure 4. After the client loads the ORB, it bounds to the application server via the Gatekeeper by calling a special bind method and passing as parameter *only* the unique service name of the application server. The client then calls the appropriate method on the application server to carry out the request. The application server will execute the client's request on the database, and return the result table as the return value of the method called.

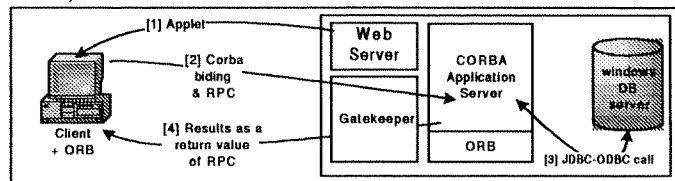


Figure 4: CORBA approach

The cost of the first query is

1. Initialization phase:
 - A. The time for the client to initialize core ORB classes.
 - B. The time for the client to bind to the application server.
2. Execution phase:
 - A. The time for the client to invoke a method on the application server passing the SQL statement as a parameter.
 - B. The time for the application server to execute the SQL, obtain and return the results to the client.

Execution phase is only required for any subsequent query.

3.2.3 The Applet JDBC approach: Applets that use directly the JDBC API. In this approach, the client applet downloads a type 3 JDBC driver and uses directly the JDBC API to connect to the database. The Gateway of the type 3 driver plays the role of the middleware. We used a type 3 driver because it is the only JDBC

driver that satisfies our two design principles discussed at the beginning of this section.

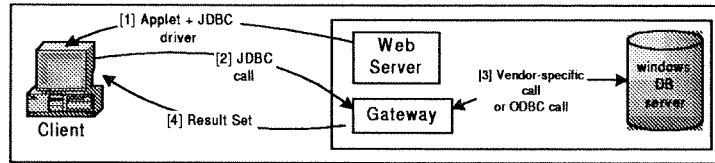


Figure 5: The Applet JDBC approach using a type 3 JDBC driver

After the client downloads the JDBC driver, it establishes database connectivity issuing JDBC calls on the Gateway, which are subsequently mapped on the database server. Figure 5 illustrates the four steps involved in the first query. Their cost is

1. Initialization phase:
 - A. The time for JDBC driver to be downloaded from the Web server and initiated by the applet.
 - B. The time for the applet to establish connection to the database though the gateway program.
2. Execution phase:
 - A. The time for the applet to issue an SQL statement to the database and obtain the results.

All subsequent queries require only the execution phase.

3.3 RPC-like approach: Java Mobile Agents (JMA).

Finally, in this subsection, we describe the approach of using mobile agents to achieve Web database connectivity, and specifically, the best of the three variants proposed in [15]. Mobile agents [3, 7] are processes capable of pausing their execution on one machine, dispatching themselves on another machine and resuming their execution on the new machine. The idea in the JMA approach is to use one or more mobile agents to implement the middleware and carry out the requests of the client. In the best variant, the results as well as subsequent queries are sent to and from the client using a message. This message passing is implemented implicitly as an RPC invocation from the client applet on the dispatched mobile agent.

For our experiments, we used *Aglets* [10], for two reasons: (a) availability of code, and (b) support for hosting mobile agents within applets without significant overhead based on our prior experience with their use. Aglets can be fired from within a special applet, called the *FijiApplet* that provides an aglet execution environment similar to the general stand-alone aglet runtime environment called the *Tahiti Server*.

In the JMA approach, the middleware consists of three components: The *DBMS-aglet*, the *(Stationary) Assistant-aglet* and the *Aglet Router*. The *DBMS-aglet* can connect to a database and submit queries. Each database server is associated with

an Assistant-aglet identified by a unique aglet ID and the URL of its Tahiti server. An Assistant-aglet provides the information necessary for a DBMS-agent to load the appropriate JDBC driver and connect to the database server. An *Aglet Router* is required to route aglets and messages, dispatched from a FijiApplet to any destination, and vice versa, because of the Java security restrictions. An aglet created within a FijiApplet is neither allowed to dispatch, nor to send a message directly to any URL other than the Web server URL.

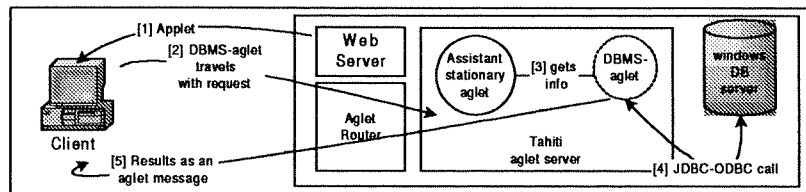


Figure 6: Mobile agents approach configuration (message variation)

When the user enters his first query (Figure 6), the client applet (an extension of the FijiApplet) creates a DBMS-aglet with a specific URL-based itinerary (travel plan) and the specified SQL statement. The DBMS-aglet travels through the aglet router to the database server machine. Upon its arrival, the DBMS-aglet communicates with the Assistant-aglet to retrieve information on the database and the available JDBC driver. It then loads the JDBC-ODBC driver, connects to the database server and executes the client's request. After sending the query result in a message to the client, the DBMS-aglet remains connected to the database server, waiting for a message with new requests from the client. The cost of the initial query is

1. Initialization phase:
 - A. The time for the client to create the DBMS-aglet
 - B. The time for the client to initialize the DBMS-aglet (SQL statement, itinerary, etc.)
 - C. The time for the DBMS-aglet to travel to the remote database server
 - D. The time for the DBMS-aglet to negotiate with the assistant aglet
 - E. The time for the DBMS-aglet to establish connection with the database
2. Execution phase:
 - A. The time for the DBMS-aglet to query the database and send the results to the client using a message.

All subsequent requests required only one message from the client to DBMS-aglet, which includes the new SQL statement, plus the execution phase.

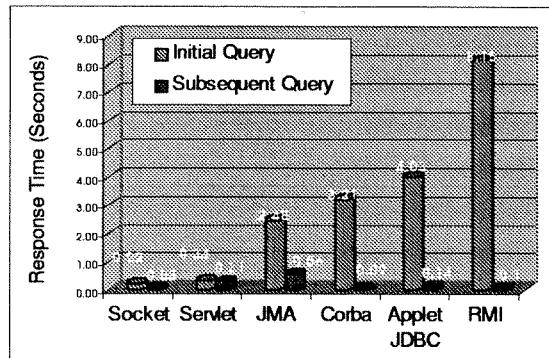
4 Performance Evaluation

We conducted two sets of experiments to evaluate the ability of each approach to support (1) *small interactions* that typically involve a small size of query results (128 bytes), and (2) *heavy cooperation* that involve a wide range of query results.

Given our interest to support both mobile clients and clients over a wide-area network with relatively slow communication links (limited bandwidth), we conducted our experiments on a wireless 1.2Mbps LAN of Pentium PCs. We used Netscape Navigator v4.6 as the Web client's Java-enabled browser. For each approach, a sufficient number of runs were performed to obtain statistically significant results.

4.1 Small Interactions

We measured the response time (a) of the first query and (b) of subsequent queries (Graph 1). Short-duration interactions consist of a single query as opposed to long-duration ones. The first query differs from the subsequent ones because it incurs the overhead of establishing the connection between the client and the remote database.



Graph 1. Performance of all approaches for 128 bytes result size

For the first query (short-duration interactions), the non-RPC approaches have by far the lowest response time. This can be explained by the fact that their initialization phase does not engage any special package loading by the client. Compared to the Socket approach, the Servlet approach performs slightly worse because (a) the communication between the client and the servlet is marshaled by the Web server, and (b) by executing as a Web server thread, the servlet receives less CPU time than the socket application server. Thus, servlets respond slower to requests and take more time to assemble the query results.

From the other approaches, the JMA approach offers the best performance for a single query. Significant part of its cost (around 2 sec) is due to the process of dispatching the DBMS-aglet from the client applet to the aglet router on the Web server and from there to the database server. In the case of the CORBA approach, the first query is slightly more expensive than the one in the JMA approach because of the overhead of initializing the necessary ORB classes and the binding to the application server. This overhead is quite significant (around 3.20 sec). Following the CORBA approach is the Java JDBC approach in which the response time of the

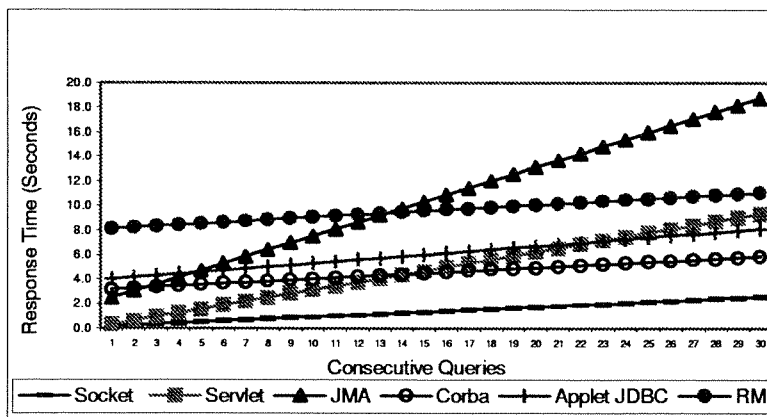
first query is increased by a considerable amount of time by the downloading of the JDBC driver.

To our surprise, the RMI approach performs by far the worst for the first query. We expected the RMI approach to exhibit better performance because, as opposed to the other RPC approaches, it does not involve the loading of any specific package. The only way to explain this is to attribute the increased response time to the interpreted method of RMI calls when binding the client applet to the application server. CORBA compilers create hard-coded encoding/decoding routines for marshaling of objects used as RPC parameters, whereas RMI uses object serialization in an introspective manner. This means that (a) RMI encodes additional class information for each object passed as a RPC parameter, and (b) marshaling is done in an interpreted fashion. Consequently, RMI remote calls are more demanding in terms of CPU time and size of code transmitted, a fact that we observed in all our experiments.

For subsequent queries (long-duration interactions), the performance of the CORBA and RMI approaches dramatically improves, and becomes close to the best performance exhibited by the Socket approach. The reason is that the client applet is already bound to the application server and only a remote procedure call is required to query the database. For a similar reason, the JDBC applet approach also exhibits a significant performance improvement for subsequent queries.

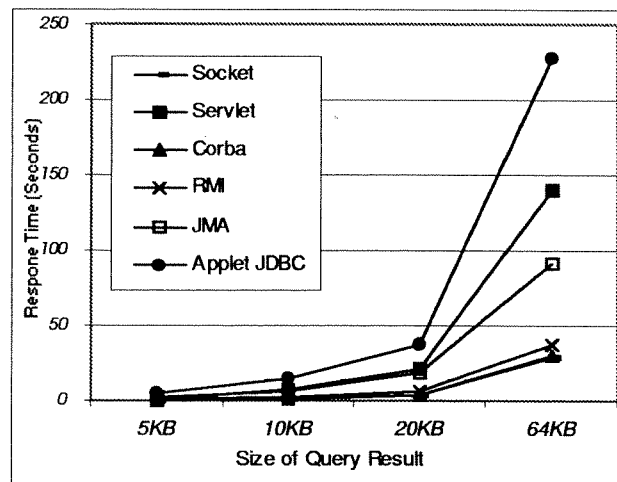
Having the DBMS-aglet already connected to the remote database and ready to process a new query, the JMA approach also improves its response time for subsequent queries. However, this response time is the worst of all other approaches. We attribute this to two reasons: (1) the two required messages to implement subsequent queries have to be routed through the aglet router, and (2) a mobile agent is not a stand-alone process and it does not receive full CPU time.

Finally, the Servlet approach improves slightly its performance although the steps for executing any query are the same. This improvement is due to the fact that any subsequent connection between the client and the Web server require less time because the URL of the Web server has already been resolved in the initial query.



Graph 2: Average performance for up to 30 consecutive queries (128 bytes of result size)

In order to better illustrate the *overall performance* of each approach, we plotted in Graph 2 the average time required by each approach for a number of consecutive queries. It is clear that the socket approach is the most efficient for both short and long interactions. This is not a surprise since all other approaches are built on top of sockets. Both the Servlet and JMA approaches scale very badly. The CORBA, JDBC applet, and RMI approaches appear to scale well, however, the RMI approach appears less attractive due to its worst performance for initial queries.



Graph 3. Subsequent Query

4.2 Heavy Cooperation

In order to evaluate heavy cooperation we adjusted the size of the query result from 5 kilobytes (95 tuples) to 64 kilobytes (1000 tuples) by changing the complexity of the SQL statement issued through the client applet. Query result size directly affects the response time in two ways: (1) in the amount of time spent for the query to execute, and (2) in the transport time for the results to reach the client. For these experiments, we also measured response times of first and subsequent queries. In both cases, each approach exhibited the similar sensitivity which is shown in Graphs 3.

The first observation is that the average response times of Java JDBC applet and JMA approaches increase exponentially with query result sizes larger than 20KB. The JDBC applet approach performs by far the worst for increased result size. This can be explained by the fact that in JDBC rows from a query result are retrieved one at a time. Specifically, to retrieve one row from the query result, the client must call

a method on a Java ResultSet object, which is mapped on the remote database server through the Gateway. Consequently, for a large size of query result, a large number of those remote calls have to take place. In that case, large query results not only increase dramatically the response time but they also increase the Internet traffic.

The bad scaling of the JMA approach can be explained in the same way as the bad performance of the Servlet approach. Both mobile agents and servlets do not execute as stand-alone processes, and therefore, they do not receive full CPU time and heavily depend on the supporting execution environment. The other RPC approaches exhibit acceptable performances (close to linear for sizes above 20KB) with the CORBA approach being slightly better. As indicated above, the implementation of RPC calls in CORBA is much faster compared to RMI's one.

5. Programmability Comparison

In this section, we compare the different approaches in terms of development effort. Our goal is to understand if there is any correlation or trade-off between performance and programming complexity. To quantify the development effort, we use the number of required system calls. The number of systems calls used in each approach is, in some sense, analogous to the number of code lines implementing each approach.

Table 1 shows the total number of system calls required for each approach. Table 1 also distinguishes between the number of system calls required to establish communication between the Web client and the middleware, and the number of calls required to submit a query and get back the results.

A first observation is that the development effort of the client is related to the level of abstraction of communication between the client and the middleware, in general, and the naming scheme used to identify the database services to establish communication, in particular. Not surprisingly, the RPC approaches involve less complex APIs, more transparent client/server communication and hence exhibit high programmability. All non-RPC approaches, including the JMA approach (the RPC-like one), require more development effort and hence have low programmability.

A second observation is that despite the fact that the JMA approach supports RPC-like communication, it exhibits the *lowest* programmability as indicated by the largest number of system calls required. Most of these system calls are used to construct, maintain and execute the URL-based itinerary.

		Socket	Servlet	CORBA	RMI	Applet JDBC	JMA
System Calls		29	25	15	12	6	29
Total Number							
Establish Connection	At the Client	7	11	2	1	3	11
	At the Middleware	11	3	8	6	0	11

Submit Query and Get Results	At the Client	3	3	1	1	3	2
	At the Middleware	8	8	4	4	0	5
Client Execution Code		6K	6K	23K	9K	50K	27K
Programmability		Low	Low	High	High	High	Low

Table 1: Programmability of the approaches

Finally, the level of programmability does not correspond to the size of the client executable code. Interestingly, the Non-RPC approaches, namely, Java Socket and Servlet, support the smallest client size (6K). On the other hand, the Applet JDBC has the largest client size of 50K: the Java applet is 6K and the JDBC driver is 46K. The JMA approach is the second most resource demanding approach after Applet JDBC with 27K: Java applet 10K, FijiApplet 10K and DBMS-Aglet 7K.

6 Conclusions and Future Work

In this experimental paper, we have implemented, evaluated, and compared all currently available Java-based approaches that support persistent Web database connectivity. Our comparison proceeded along the lines of the performance of query processing and of the programmability of each approach.

The results of our comparison showed that the CORBA approach offers high programmability and hence, is easy to develop, while its performance is comparable to the best performing approach that employs sockets. Therefore, the CORBA approach offers the best promise for the development of large Web applications, in particular, in those with long cooperative interactions involving multiple queries of varying result sizes. For short interactions, typically involving a single query, and environments with resource-starved clients, the socket and servlet approaches should be considered. These approaches yield Web client with the smallest footprint, just 6 Kbytes. Clearly, the best performance is not always achievable with high programmability and low resource requirements.

The recent advancements of the Web technology and mobile computing led to a renewed interest on mobile agents technology. Given this renewed interest, our study provided an insight to potential scalability problems with the currently available mobile agent implementations. The JMA approach cannot support interactions that require movement or exchange of large amounts of data such as large number of consecutive queries with increased size of query result. Hence, it is necessary to develop more efficient mobile agent infrastructures, if the full potential of mobile agents is to be explored. As part of our future work, we investigate the possibility of merging mobile agents and the CORBA technology in order to facilitate a scalable and efficient JMA-based Web database connectivity.

References

1. E. Anuff. *Java Sourcebook*. Wiley Publishing, 1996.
2. T. Berners-Lee and D. Connolly. Hypertext Markup Language Specification 2.0, Internet Draft, *Internet Engineering Task Force (IETF), HTML Working Group*. Available at <www.ics.uci.edu/ietf/html/html2spec.ps.gz>, June 1995.
3. D. Chess, B. Grosz, C. Harrison, D. Levine, C. Parris, and G. Tsudik. Itinerant Agents for Mobile Computing. *IEEE Personal Communications*, Vol. 2, No. 5, October 1993.
4. T. B. Downing. *Java RMI: Remote Method Invocation*. IDG Books Worldwide, 1998.
5. J. Goodwill. *Developing Java Servlets*. Sams Publishing, 1999.
6. S. P. Hadjiefthymiades and D. I. Martakos. A Generic Framework for the Development of Structured Databases on the WWW. *Fifth Int'l WWW Conference*, May 1996.
7. C. G. Harrison, D. M. Chess, A. Kershenbaum. Mobile Agents: Are they a good idea? Research Report, *IBM Research Division*, 1994.
8. G. Helmayer, G. Kappel, and S. Reich. Connecting Databases on the Web: A Taxonomy of Gateways. *Eighth Int'l DEXA Conference*, Sept. 1997.
9. H. Maurer. *Hyperwave: The Next Generation Web Solution*, Addison-Wesley, 1996.
10. IBM Japan Research Group. *Aglets Workbench*. Web site: <<http://www.trl.ibm.co.jp/aglets>>.
11. B. Jepson. *Java Database Programming*. Wiley Computer Publishing, 1997.
12. A. Lambrinidis and N. Rousopoulos. Generating dynamic content at database-backed web server: cgi-bin vs mod_perl. *Sigmod Record*, March 2000.
13. Object Management Group. The Common Object Request Broker: Architecture and specification. February 1998.
14. R. Orfali, D. Harkley. *Client Server Programming with Java and CORBA*. Second Edition. Wiley Publishing, 1998.
15. S. Papastavrou, G. Samaras, and E. Pitoura. Mobile Agents for WWW Distributed Database Access. *Fourteenth IEEE Int'l Conference on Data Engineering*, Feb. 1999.
16. Sun Microsystems Inc., Java Development Kit, <<http://java.sun.com/jdk>>.
17. Sun Microsystems Inc. Java Sockets Documentation, <<http://java.sun.com/docs>>.
18. Sun Microsystems Inc., JDBC drivers, <<http://java.sun.com/products/jdbc/drivers.html>>.
19. Visibroker for Java: Programmer's Guide, V.3.0. Borland, <<http://www.inprise.com/visibroker>>.

A Survey on the Java-based Approaches for Web Database Connectivity¹

Stavros Papastavrou², Panos K. Chrysanthis², George Samaras³, Evaggelia Pitoura⁴

Abstract

The undeniable popularity of the web makes the efficient accessing of distributed databases from web clients an important topic. Various methods for web database integration have been proposed but recently there is an increasing interest on those based on Java-based ones. This is due to the inherent advantages of Java, which supports platform independence and secure program execution, and produces a small size of compiled code. In this experimental paper, we evaluate all currently available Java-based approaches. These include Java applets, Java Sockets, Servlets, Remote Method Invocation, CORBA, and Mobile Agents. To this end, we implemented a Web client accessing a remote database using each of these approaches and compared their behavior along the following important parameters: (1) *performance* expressed in terms of response time under different loads, (2) *transparency of communication* expressed in terms of complexity of networking API, and (3) *extensibility* expressed in terms of ease of adding new components.

Keywords: Distributed Databases, WWW, Java Mobile Agents, Distributed Objects, CORBA

I. INTRODUCTION

Providing efficient access to distributed databases from Web clients using a Web browser [2] is crucial for the emerging database applications such as E-Commerce. Several methods for Web database connectivity and integration have been proposed such as CGI scripts, active servers pages, server side include, databases speaking http, external viewers or plug-ins, proxy-based, and HyperWave [4]. However, there is an increasing interest in those that are Java-based due to the inherent advantages of Java [1], namely, platform independence support, secure program execution, and production of a small size of compiled code.

Several Java-based methods are currently available for Web database integration but in the best of our knowledge, there is no quantitative comparison of them. This experimental paper contributes such a comparison. Specifically, it evaluates six approaches, namely, *Java JDBC applet*, *Java Sockets* [7], *Servlet* [7], *Remote Method Invocation (RMI)* [7], *CORBA* [9], and *Java mobile agents (JMA)* [3]. Each approach differs in the way the client establishes connection with remote database servers.

For our evaluation, we used each of these approaches to implement a Web client accessing a remote database and

compared their behavior along the following important parameters: (1) *performance* expressed in terms of response time under different loads, (2) *transparency of communication* expressed in terms of complexity of networking API, and (3) *extensibility* expressed in terms of ease of adding new components. Further, we characterized these approaches in terms of the total development effort based on lines of code at both the client and the server side in conjunction with the two latter parameters, namely, transparency and extensibility.

In the next section, we briefly describe our testbed. In Sections III to IV, we elaborate on the characteristics of each approach when comparing them along the dimensions of communication transparency and extensibility. In section VI, we present our performance evaluation results.

II. EXPERIMENTAL TESTBED

Two design principles were adopted in the selection of the various components during the development of the testbed. First, our Web clients should be lean with the purpose of allowing fast downloads and therefore increasing support for wireless and mobile clients. Second, no a-priori configuration of the Web client should be necessary to run the experiments in order to maintain portability, and therefore support arbitrary clients.

For every approach, our Web client program was a Java applet, which was installed on a Web server machine along with an html page. Every experiment was initiated by first pointing to the html page from a remote client computer (Figure 1). After the Java applet was initialized at the client computer, queries were issued through the applet's GUI and executed at the remote database server. Our remote database server, a 3-table relational Microsoft Access database, was installed on the same machine with the Web server. The communication between the server and the client computer was a wireless LAN at 1.2Mbps.

In all cases, the client establishes Web database connectivity through a middleware program typically running on the Web server machine. For the Java JDBC applet, we used a type 3 JDBC driver in accordance to our design criteria. In this case, the middleware corresponds to the middle-tier gateway of the type 3 JDBC driver [8]. In the case of JMA, the middleware is a local stationary agent that provides the information necessary for a mobile agent to load the appropriate JDBC driver and connect to the database server. In our experiments, we used DBMS-aglets [6]. In all other cases, we developed the middleware program, which plays the role of an application server that uses a JDBC-ODBC bridge driver to connect to the database server. For more details, see [5].

¹ This work was partially supported by NSF grants IRI-9502091 and IIS-9812532, and AFOSR award F49620-98-1-043.

² Computer Science Dept., Univ. of Pittsburgh, PA 15260, USA.

³ Computer Science Dept., Univ. of Cyprus, Nicosia, Cyprus.

⁴ Computer Science Dept., Univ. of Ioannina, Ioannina, Greece.

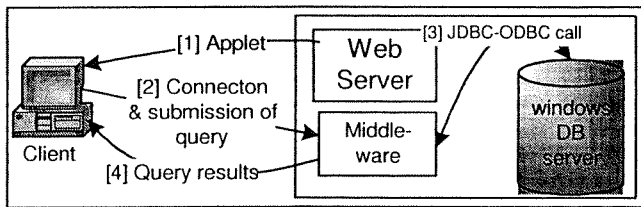


Figure 1: Basic configuration

III. TRANSPARENCY OF COMMUNICATION

The dimension of the transparency of communication deals with the level of abstraction of communication between the client and the server side, in other words, between the client and the middleware program. The approaches can be broadly classified as (1) *non-RPC* ones, that do not support any clear remote method invocation mechanism, and (2) *RPC* ones with clear remote method invocation semantics. The Java Sockets and Servlets are non-RPC approaches in which information between the client and the middleware program is exchanged using streams of data. Java JDDC applets, RMI, CORBA and JMA are all RPC approaches.

Table 1 compares the transparency of communications of the discussed approaches based on the complexity of the networking API employed by each approach. Clearly, the RPC approaches involve less complex networking APIs and hence more transparent client/server communication.

The CORBA approach offers the *highest* level of communication transparency since it requires knowledge of neither URLs nor port numbers to establish database connectivity. It only requires the reference name with which the application server was registered at the server site. One level lower (*high*) is the RMI approach, which requires the URL of the database server along with the reference name of the application server. Similar to the RMI approach, the Java JDBC Applet approach requires the URL of the database server, and a data source name, which identifies the database itself. In this same level is the JMA approach. Mobile agents identify remote host machines with their URL, and interact with other agents using their unique identifiers. One level below (*low*) is the Servlet approach requiring a URL, the servlet name, and the type of operation to be executed by the particular servlet. Finally, as expected, the approach with the *lowest* communication transparency is the socket approach, which requires knowledge of the IP and of the port number of the application server.

IV. EXTENSIBILITY

We define *extensibility* to be: (a) the ability of adding new components to an approach (e.g., a new application server object attached to a local or a remote database) and binding them with the existing ones at the server site; and (b) the level of modifications needed at the client part that will enable the client to utilize newly added components. We classified the various approach in terms of extensibility as highest, high, average and low (Table 1).

The approaches with the *highest* degree of extensibility are the CORBA and JMA. In the one based on CORBA, the application server and the client applet can bind to a newly

	Applet JDBC	Socket	Servlet	RMI	CORBA	JMA
Transparency	High	Lowest	Low	High	Highest	High
Extensibility	High	Lowest	High	Average	Highest	Highest
Code Size	4c	22c	16c	9c	10c	30c
Total effort	Lowest	Highest	High	Low	Lowest	Low

Table 1: Effort of development

added component by only using its reference name. As opposed to other approaches, new components need not be necessarily located at the Web server machine in order for the client to bind to them.

The JMA approach is inherently very extensible since mobile agents were designed to autonomously collect information and exploit any newly added servers in order to complete their execution plan. Moreover, the Web client need not be aware of the existence of new servers.

The *high* extensibility of the JDBC applet approach is due to the type 3 JDBC driver used. Of all the JDBC drivers, type 3 drivers are the most extensible because of their middle-tier gateway that maps client applet's database requests to any local or remote database calls. The Web client only needs to name the newly added databases.

The servlet approach also offers *high* extensibility. Servlets execute in the context of the Web server and can call (explicitly) other servlets within the same context. This means new servlets can be added without any Web client modification. The client applet can also call explicitly a new servlet using as a reference the URL, the new servlet's name and the type of operation that must be executed by it.

Because of their similarities, one might have expected that RMI and CORBA approaches would exhibit the same degree of extensibility. However, compared to the CORBA approach, the RMI approach is much less extensible for three reasons. First, new components must be written only in Java. Second, new components are identified, besides of their reference name, with an additional URL. Lastly, the client applet cannot bind to new components that reside on URLs other than the Web server. Clearly, RMI also offers lower extensibility than the Java JDBC applet approach.

Finally, the approach with the *lowest* degree of extensibility is the socket one. For any new component, a new socket must be created, bound and managed either at the side of the application server or at the Web client.

V. EFFORT OF DEVELOPMENT

The effort of development basically combines the dimensions of transparency of communications and extensibility, and quantifies them in terms of lines of code. In Table 1, the lines of code for each approach are normalized with a constant *C*.

The approaches with the lowest effort of programming are the Java JDBC applet and the CORBA. The applet approach combines the fewer relative lines of code, high level of network transparency and an average extensibility, while the CORBA approach offers the highest transparency and extensibility with relative small code size.

The high extensibility and transparency of the JMA approach comes with a premium in terms of lines of code. Mobile agents involved significant programming. The RMI approach is the opposite of the JMA one. It requires a

relatively low number of lines of code and offers average extensibility.

The Servlet and Socket approaches involve the most effort of development, given their large number of lines of code and their low transparency and extensibility.

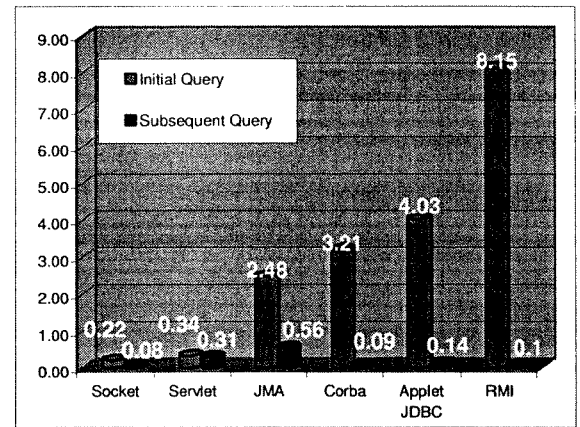
VI. PERFORMANCE EVALUATION

For our performance evaluation, we measured the average response time for our Web client (a) to query the remote database for the first time, (b) to query the remote database for a number of subsequent times. Querying the remote database for the first time differs from subsequent queries because the first query involves the additional overhead of establishing the communication link between the client and the remote database.

The other significant issue that we considered in our experiments is the size of the query result. Query result size directly affects the response time in two ways. First, in the amount of time spent for the query to execute, and second, in the transport time for the results to reach the client. We adjusted the size of the query result by changing the complexity of the SQL statement issued through the client applet. For our experiments we measured average response times for a wide range of query result sizes, beginning from 128 bytes (8 tuples) up to 64 kilobytes (1000 tuples). For each approach, a sufficient number of runs were performed in order to obtain statistically significant results [11]. Below we first focus on the experiments for small query results (128 bytes) and then discuss how the response time of the different approaches is affected by the query size.

Graph 1 shows the average response time for the initial and subsequent queries in each approach. For the initial query, the non-RPC approaches have by far the lowest response time. This can be explained by the fact that their initialization phase does not engage any special package loading or handling by the client. Compared to the Socket approach, the Servlet approach performs slightly worse because (a) the communication between the client and the servlet is marshaled by the Web server, and (b) by executing as a Web server thread, the servlet receives less CPU time than the socket application server. Thus, servlets respond slower to requests and require more time to assemble and return the query results.

From the RPC approaches, the JMA approach offers the best performance for a single (initial) query. Significant part of its cost (around 2 seconds) is due to the process of dispatching the DBMS-aglet from the client applet to the aglet router on the Web server and from there to the database server. In the case of the CORBA approach, the first query is slightly more expensive than the one in the JMA approach because of the overhead of initializing the necessary ORB classes and the binding to the application server. This overhead is quite significant (around 3.20 seconds) which can be clearly seen by comparing the response time of the initial and subsequent queries. Following the CORBA approach is the Java JDBC approach in which the response time of the initial query is increased by a considerable amount of time by the downloading of the JDBC driver from the Web server.



Graph 1: Performance of all approaches for initial and subsequent query (128 bytes result size)

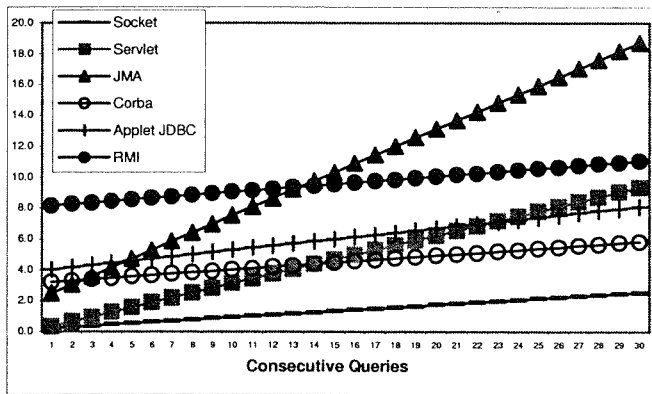
To our surprise, the RMI approach performs by far the worst for the initial query. We expected the RMI approach to exhibit better performance because, as opposed to the other RPC approaches, it does not involve the loading of any specific package during initialization time. The only way to explain this behavior is to attribute the increased response time to the interpreted method of RMI calls when binding the client applet to the application server.

For subsequent queries, the performance of the CORBA and RMI approaches dramatically improves, and becomes close to the best performance exhibited by the Socket approach. The reason is that the client applet is already bound to the remote application server and only a remote procedure call on the application server is required to query the database. For a similar reason, the Java JDBC applet approach also exhibits a significant performance improvement for subsequent queries - the JDBC driver is already downloaded and initialized at the client applet. Having the DBMS-aglet already connected to the remote database and ready to process a new query on behalf of the client applet, the JMA approach also improves its response time for subsequent queries. However, this response time is the worst from all the other approaches. We attribute this to two reasons. First, the two required messages to implement subsequent queries have to be routed through the aglet router, and second, a mobile agent is not a stand-alone process and it does not receive full CPU time.

On the other hand, the Java Servlet approach improves only slightly its performance because the steps for executing a subsequent query do not differ from the ones for the initial query. The minor improvement is due to the fact that any subsequent URL connections from the client applet to the Web server require less time since the address of the Web has already been resolved in the initial query.

In order to better illustrate the *scalability* of each approach, we plotted in Graph 2 the average time required by each approach to query the database for a number of consecutive requests using the formula: For n consecutive queries, the average time required is the sum of (a) the average response time for one initial query, and (b) $n-1$ times the average response time for a subsequent query.

As shown in Graph 2, the socket approach is the most efficient for any number of consecutive queries. Despite its



Graph 2: Average performance for up to 30 consecutive queries (128 bytes of result size)

good performance for initial queries, the Servlet approach does not scale well since the response time for subsequent queries almost matches the response time for initial queries. Likewise, the JMA approach scales very badly given that its response time for subsequent queries is the worst of all the approaches. The CORBA, Java JDBC applet, and RMI approaches appear to scale well, however, the RMI approach appears less attractive due to its worst performance of all the approaches for initial queries.

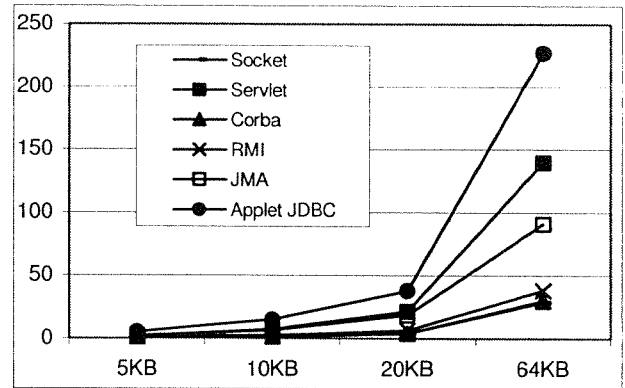
Graph 3 illustrates the sensitivity of each approach to the size of query results. Due to space limitations, we show here only the results for subsequent queries. The results for initial queries are similar.

The first striking observation is that the response time of the Java JDBC applet and JMA approaches increases exponentially with query result sizes larger than 20KB. The Java JDBC applet approach performs by far the worst for increased result size. This can be explained by the fact that in JDBC rows from a query result are retrieved one at a time. Specifically, to retrieve one row from the query result, the client must call a method on a Java ResultSet object, which is mapped on the remote database server through the Gateway. Consequently, for a large size of query result, a large number of those remote calls have to take place. In that case, large query results not only increase dramatically the response time but they also increase the Internet traffic.

The bad scaling of the JMA approach can be explained in the same way as the bad performance of the Servlet approach. Both mobile agents and servlets do not execute as stand-alone processes, and therefore, they do not receive full CPU time and heavily depend on the supporting execution environment. The other RPC approaches exhibit acceptable performances (close to linear for sizes above 20KB) with the CORBA approach being slightly better. As indicated above, the implementation of RPC calls in CORBA is much faster compared to RMI's one.

VII. CONCLUSIONS

In this experimental paper, we have implemented, evaluated, and compared all currently available Java-based approaches for Web database connectivity. Our comparison was based on the performance of query processing, the transparency of communication and extensibility.



Graph 3: Subsequent Query

The results of our comparison showed that the CORBA approach is the most transparent to communication, extensible and easy to develop, while its performance is comparable to the best performing approach that employs sockets. Hence, it offers the best promise for the development of large Web applications.

In our study, we confirmed the desirable properties of the emerging mobile agents technology, that is, of high extensibility and transparency at a relatively low development effort. But, at the same time, our study provided an insight to potential scalability problems with the currently available mobile agent implementations. The JMA approach cannot support interactions that require movement or exchange of large amounts of data such as large number of consecutive queries with increased size of query result. Hence, it is necessary to develop more efficient mobile agent infrastructures, if the full potential of mobile agents is to be explored. As part of our future work, we investigate the possibility of merging mobile agents and the CORBA technology in order to facilitate a scalable and efficient Web database connectivity.

REFERENCES

- [1] E. Anuff. *Java Sourcebook*. Wiley Publishing, 1996.
- [2] S. P. Hadjiefthymiades and D. I. Martakos. A Generic Framework for the Development of Structured Databases on the WWW. *Fifth Int'l WWW Conference*, May 1996.
- [3] C. G. Harrison, D. M. Chessm, A. Kershenbaum. Mobile Agents: Are they a good idea? Research Report, IBM Research Division, 1994.
- [4] G. Helmayer, G. Kappel, and S. Reich. Connecting Databases on the Web: A Taxonomy of Gateways. *Eighth Int'l DEXA Conference*, Sept. 1997.
- [5] S. Papastavrou, P.K. Chrysanthi, G. Samaras, and E. Pitoura. An Evaluation of the Java-based Approaches for Web Database Connectivity. CSD Technical Report. University of Pittsburgh, Mar.2000.
- [6] S. Papastavrou, G. Samaras, and E. Pitoura. Mobile Agents for WWW Distributed Database Access. *Fourteenth IEEE Int'l Conference on Data Engineering*, Feb. 1999.
- [7] Sun Microsystems Inc., Java Development Kit, <<http://java.sun.com/jdk>>.
- [8] Sun Microsystems Inc., *JDBC drivers*, <<http://java.sun.com/products/jdbc/drivers.html>>.
- [9] Visibroker for Java: Programmer's Guide, Version 3.0. Borland, <<http://www.visigenic.com>>.